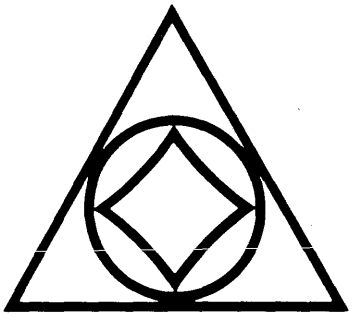


AFIPS
CONFERENCE PROCEEDINGS
VOLUME 23

1963
SPRING JOINT COMPUTER CONFERENCE

1963
SPARTAN BOOKS, INC.
Baltimore, Md.
CLEAVER-HUME PRESS
London

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1963 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number: 55–44701

Sole Distributors in Great Britain, the British Commonwealth and the Continent of Europe:

CLEAVER-HUME PRESS
10-15 St. Martins Street
London W. C. 2

LIST OF JOINT COMPUTER CONFERENCES

1. 1951 Joint AIEE-IRE Computer Conference, Philadelphia, December 1951
2. 1952 Joint AIEE-IRE-ACM Computer Conference, New York, December 1952
3. 1953 Western Computer Conference, Los Angeles, February 1953
4. 1953 Eastern Joint Computer Conference, Washington, December 1953
5. 1954 Western Computer Conference, Los Angeles, February 1954
6. 1954 Eastern Joint Computer Conference, Philadelphia, December 1954
7. 1955 Western Joint Computer Conference, Los Angeles, March 1955
8. 1955 Eastern Joint Computer Conference, Boston, November 1955
9. 1956 Western Joint Computer Conference, San Francisco, February 1956
10. 1956 Eastern Joint Computer Conference, New York, December 1956
11. 1957 Western Joint Computer Conference, Los Angeles, February 1957
12. 1957 Eastern Joint Computer Conference, Washington, December 1957
13. 1958 Western Joint Computer Conference, Los Angeles, May 1958
14. 1958 Eastern Joint Computer Conference, Philadelphia, December 1958
15. 1959 Western Joint Computer Conference, San Francisco, March 1959
16. 1959 Eastern Joint Computer Conference, Boston, December 1959
17. 1960 Western Joint Computer Conference, San Francisco, May 1960
18. 1960 Eastern Joint Computer Conference, New York, December 1960
19. 1961 Western Joint Computer Conference, Los Angeles, May 1961
20. 1961 Eastern Joint Computer Conference, Washington, December 1961
21. 1962 Spring Joint Computer Conference, San Francisco, May 1962
22. 1962 Fall Joint Computer Conference, Philadelphia, December 1962
23. 1963 Spring Joint Computer Conference, Detroit, May, 1963

Conferences 1 to 19 were sponsored by the National Joint Computer Committee, predecessor of AFIPS. Back copies of the proceedings of these conferences may be obtained, if available, from:

- Association for Computing Machinery, 14 E. 69th St., New York 21, N. Y.
- American Institute of Electrical Engineers, 345 E. 47th St., New York 17, N. Y.
- Institute of Radio Engineers, 1 E. 79th St., New York 21, N. Y.

Conferences 20 and up are sponsored by AFIPS. Copies of AFIPS Conference Proceedings may be ordered from the publishers as available at the prices indicated below. Members of societies affiliated with AFIPS may obtain copies at the special "Member Price" shown.

Volume	*List Price*	*Member Price*	*Publisher*
20	$12.00	$7.00	Macmillan Co., 60 Fifth Ave., New York 11, N. Y.
21	6.00	6.00	National Press, 850 Hansen Way, Palo Alto, Calif.
22	8.00	4.00	Spartan Books, Inc., 301 N. Charles St., Baltimore 1, Md.
23	10.00	5.00	Spartan Books, Inc., 301 N. Charles St., Baltimore 1, Md.

NOTICE TO LIBRARIANS

This volume (23) continues the Joint Computer Conference Proceedings (LC55–44701) as indicated in the above table. It is suggested that the series be filed under AFIPS and cross referenced as necessary to the Eastern, Western, Spring, and Fall Joint Computer Conferences.

CONTENTS

PREFACE

This volume contains the full text of the thirty-seven technical papers selected for presentation and discussion at the 1963 Spring Joint Computer Conference. It also includes a summary of one of the special panel discussions. Thus it provides a permanent record of the more formal side of the Conference.

The material herein represents a broad cross-section of activity in computer and information-processing technology, as of early 1963. In organizing the program for the Conference, general areas of interest were tentatively established for each session, and these were used as guides in selecting and grouping papers. On the whole, however, no real constraints were imposed as to subject matter.

Indicative of the scope are three papers devoted specifically to critical analyses of the current state-of-the-art. In other papers, the subjects discussed range from basic concepts to practical applications, in both hardware and software, reflecting the ever-broadening impact of information processing on modern society.

This volume is the product of much hard work on the part of the authors who prepared the individual papers. We are indeed grateful for these contributions—clearly the backbone of any highly technical Conference such as this. It is also a pleasure to acknowledge the contributions of the many others—well over a hundred—who helped organize the Conference and who participated as session chairmen, panel moderators, panelists, and reviewers.

E. CALVIN JOHNSON
General Chairman
1963 Spring Joint Computer Conference

DETERMINING FASTEST ROUTES USING FIXED SCHEDULES

Dr. Bernard M. Levin and Mr. Stephen Hedetniemi
National Bureau of Standards
Washington 25, D. C.

INTRODUCTION

An interesting problem that is amenable to solution by digital computer is posed by the following questions. How late can a shipment be detained at city A so that it arrives at city B by a given time? By what route should it be sent? The available routes consist of those provided by scheduled common carriers such as the airlines.

In some situations, no single scheduled carrier trip satisfactorily connects the two cities involved. In such a case it might be necessary to use two vehicles and to transfer the mail between the two at a third city. Conceivably, it might be necessary to use three or more vehicles and two or more transfer points.

The problem may have more meaning if it is posed by the following more personal questions. How late can I stay in my home town and still get to an appointment in another city on time? What route should I take?

An important application of a solution to this problem can be found in the Post Office Department. The Department tries to process mail received in the afternoon so that it will be delivered the following morning at distant cities.

This problem has been studied at the National Bureau of Standards under the sponsorship of the Post Office Department. This paper describes and discusses some solutions that have been obtained. These solutions are related to and stem from published literature regarding the Shortest Route Problem.

THE PROBLEM

Relationship to the Shortest Route Problem

Given a network of points and lines, the latter numbered by the distance between the points they connect, it may be of interest to know the shortest path between any two points. This is the shortest route problem and many solutions to this problem appear in the literature.[1, 3, 5, 7]

If the points correspond to cities and the values of the lines correspond to the travel times between the cities, solutions to the shortest route problem yield the route with the least time in transit. These solutions do not necessarily give optimum answers to the questions posed in the introduction. For example, suppose we live in Boston, Mass. and must get to Binghamton, N.Y., for a luncheon meeting. There is a plane that leaves Boston at 6:30 a.m. and which arrives in Binghamton at 8:37—a total flight time of 127 minutes. Getting up in time to catch a 6:30 a.m. plane is not a very satisfying thought, so we investigate the possibility of going another way. We find we can take a plane from Boston to New York's Idlewild Airport, leaving Boston at 7:45 a.m. and arriving at Idlewild at 8:38. We can then transfer to another plane leaving Idlewild at 9:30 and arriving Binghamton at 10:37. This routing requires 172 minutes from the time the first plane is scheduled to leave Boston to the time the second plane arrives at Binghamton; 52 minutes of the elapsed time is involved in transferring from the first plane to the second. This

is not the shortest route, but it may be the desired one.

Factors to be Considered

If our only consideration is to leave Boston as late as possible in order to make our luncheon appointment, then the routing by way of Idlewild is superior because it leaves 75 minutes later than the direct flight. There are, however, other factors that must be taken into consideration in deciding which routing is superior, in addition to flight time and departure time.

1. *Cost:* It costs $21.80 to fly directly from Boston to Binghamton. It costs $28.80 to fly from Boston to Idlewild to Binghamton. In other words, we must pay seven dollars plus tax for the extra 75 minutes of sleep.

2. *Reliability:* The routing by way of Idlewild allows 52 minutes for transferring from the Boston-Idlewild plane to the Idlewild-Binghamton plane. If the Boston-Idlewild plane were late, the connection might be missed, causing us to be stranded at Idlewild and, therefore, to miss our luncheon. And further, if the Idlewild-Binghamton plane were late, we might also miss our luncheon. Also, any scheduled airline flight is subject to cancellation due to weather or equipment problems. The direct flight from Boston to Binghamton involves only one plane and therefore such problems are less likely to occur than in the two plane routing by way of Idlewild. In other words, to get 75 minutes more sleep, we would increase the probability of not arriving by the required time.

3. *Transfer time:* We have considered the problem of missing connections due to the late arrival of the first plane in a two-plane routing. Even if the first plane is on time, some time must be allowed for transferring from it to another plane. This time varies according to the size of the airport and according to whether or not the transfer is between two planes of the same airline or between planes of two different airlines. The recommended minimum time for transferring at Idlewild from the Boston-Idlewild flight to the Idlewild-Binghamton flight is 30 minutes, which is less than the available 52 minutes. In addition to the 7:45 a.m. flight, there is a plane from Boston to Idlewild leaving Boston at 8:30 a.m. arriving at Idlewild at 9:22 a.m., only 8 minutes before the departure from Idlewild to Binghamton. We would not select this flight because eight minutes is not sufficient transfer time.

In summary, we have stated five factors that are of importance in selecting routes: (1) departure time, (2) cost, (3) reliability, (4) transfer time, and (5) arrival time. The only required property of the arrival time is that it be before a specified time.

PRELIMINARY SOLUTIONS

Selection of Flights Along a Known Route

A digital computer can be programmed to select flights along a known path. For example, the known path could be Boston to Idlewild to Binghamton. There are several flights each day between both pairs of airports. The computer can be programmed to select the set of flights that best meets the criteria discussed in the previous section. Only the criterion of time will be discussed in this section.

To simplify matters, the procedures in this paper will be described by means of terminology peculiar to airline flights. The procedures can be used for air transportation, surface transportation, and for combinations of surface-air transportation.

The selection of the flights is not as simple a task as it may at first appear. One solution would be to check all possible combinations of flights. While this is simple conceptually, it involves an inefficient use of the computer. Another obvious approach would be to select a desirable departure time from Boston and choose the flight to Idlewild with the departure time closest to this desired time. The first flight leaving Idlewild for Binghamton after the arrival of the selected Boston-to-Idlewild flight would be the selected second link flight. This solution is efficient but has two important shortcomings which are easy to overcome:

1. It overlooks some problems involved in making transfers.
2. It is not designed to answer one of the specific questions asked, namely, that of leaving as late as possible.

In making transfers from one flight to another, it is necessary to consider whether or not the transfer is between two planes of the same airline or between two planes of different airlines. A transfer between two planes of the same airline is called an intra-line transfer; a transfer between planes of different airlines is

called an inter-line transfer. The minimum time allowed to make a transfer is usually less in the first case than in the latter. There is no problem in having two minimum transfer times for each airport: one for intra-line transfers and one for inter-line transfers. The problem arises when the first flight to arrive requires an inter-line transfer and a second flight, which arrives soon afterward, involves an intra-line transfer. It is possible that the earlier flight cannot make connections with the plane which departs at the desired time, while the later flight can. For example, suppose there is a flight from city A to city B leaving at 7:00 and arriving at 8:00 on airline one. Suppose there is also a flight between the same two cities leaving at 7:05 and arriving at 8:05 on airline two, and there is a flight on airline two departing city B for city C at 8:45. If intra-line transfers require 30 minutes and inter-line transfers require 60 minutes, then the flight which arrives at 8:05 can make connections, while the 8:00 flight cannot. The solution to this problem of transfer times is to consider as potential first-link flights all flights whose arrival time at city B is within X minutes after the earliest arrival time at city B, where X is the difference between the inter-line and intra-line transfer times at city B. In cases of routes having more than one transfer point, this factor must be considered at each transfer point.

The problem posed in Section I was to select a routing that permitted the shipment to be detained at the originating city as late as possible. The solution described so far uses a prescribed earliest departure time and, therefore, does not really solve the problem posed. This can easily be corrected by tracing the route backwards. The last leg of the trip would be selected first, based on its arrival and departure times, and so forth. The basic nature of the solution would be unchanged. In overcoming the transfer problem described in the previous paragraph, we would consider as potential last link flights all flights whose departure time at city B is within X minutes before the latest departure time at city B.

Selection of Promising Routes

The backward tracing of the path with proper correction for the transfer problem will select the optimum path on a basis of the criterion of latest departure. However, it requires the routing to be predetermined. In order to find the best set of flights, it will often be necessary to trace out numerous routes. These routes can be determined by another computer program or by someone making decisions regarding the routes to be tested. An algorithm has been developed which finds the N shortest routes between two given cities where $N_1 < N < N_2$, and N_1 and N_2 are variables. This algorithm considers only the travel time among the cities and does not consider delays involved in transferring. It will be the subject of a separate paper now being prepared. (It would have been possible also to have used other procedures for finding the N shortest routes.)[2, 4] For each of the N routes, a set of flights is selected and compared with the sets of flights for the other routes, and the set of flights with the latest departure time selected. A computer program has been written which selects the N shortest routes neglecting transfer times, selects a set of flights for each route considering transfer times, and rank orders them on the basis of latest departure time. This program uses as input data IBM cards containing the name of the airline, the flight number, the departure airport and time, and the arrival airport and time. It can handle 2,000 trip segments and 45 transfer points. To save space in the computer memory, a trip that has several stops is considered as a series of nonstop trip segments. For example, a plane trip from Boston to New York to Washington is treated as two trip segments: Boston to New York and New York to Washington.

The program was written in FORTRAN for the IBM 704 and 7090 computers. On the IBM 7090, it took two minutes to find paths between 30 pairs of cities. This involved determining and tracing 265 paths.

There is no assurance that the best route will be one of the N shortest routes as determined by the algorithm. All of the N shortest routes may involve poor connections at the transfer points while a longer routing may involve good connections at the transfer points. This is a weakness of the procedures. Also, cost is not considered.

Until now, the only consideration given to the arrival time has been that it be before the prescribed arrival time. After the latest possible

departure time at the city of origin is determined, consideration can be given to selecting the earliest arrival time. The airline passenger may wish to depart as late as possible, but he would prefer having idle time in the city of destination rather than at a transfer point. In the same way, we may want to have the departure of the last dispatch of mail as late as possible. (Mail ready for an early dispatch would be sent on the early one.) However, once the dispatch time is set, it would be desirable to get the mail to its destination as early as possible so that there would be more time to process it at the destination post office. This can be accomplished by retracing the path in a forward direction after the departure time has been determined by the backward tracing. For example, if we wish to leave Minneapolis and arrive at Cleveland by 7:00 p.m., we would find, by tracing backwards, that we could take a 3:45 plane from Chicago, arriving in Cleveland at 6:20. We could take a 1:45 plane from Minneapolis to Chicago and connect with the 3:45 plane. Tracing forward, we find we could take the 1:45 plane from Minneapolis and make connections in Chicago with a plane arriving at Cleveland at 6:05, fifteen minutes earlier than the trip selected by the backward tracing.

Selection of Optimum Routes

A computer program has been written that does not have the limitations just described. It always finds the fastest route; it includes cost as a factor; and it does not require both a forward and backward tracing. It is based on the algorithm presented in the Appendix (with modifications indicated below) and is similar to a procedure suggested by Minty[5]. The basic idea is to find the best direct flights given a starting time from the originating airport to all other airports being considered. The direct flights are the links of the network. Each of these selected direct flights is considered as a possible first link of a two-link route to each of the other airports. The airport at which the direct flight lands is considered as a potential transfer point. Every flight leaving that airport is paired with the direct flight. Each resulting two-link route is compared with the best previously determined route to the arrival city of the two-link route. If it is better, it is stored in place of the previously determined best. After all the stored one-link routes are tested, the stored two-link routes are tested to see if they can be used as the first two links of a usable three-link route. Whenever the three-link route is better than the previously stored route, it is stored in place of the previous best. This process is continued until for some m, all of the m-link stored routes are tested as the first m links of a route having $m+1$ links, and none of the $(m+1)$-link routes warrant retention. This algorithm is similar to the "Moore Algorithm"[4].

The transfer problem described in the previous section arises here also. It is solved by storing more than one route between a pair of cities whenever conditions warrant, and using each of the stored routes to see if it can be the first part of a route to another city. The criteria for storing additional routes are the same as before. That is, consider as potential first-link flights all sets of flights along any route whose arrival time at city B is within X minutes after the earliest arrival time found so far at city B, where X is the difference between inter-line and intra-line transfer times at city B.

It should be noted that this solution finds routes not only from the origin city to the destination city but also from the origin city to all other cities. Hence, some routes are retained only because they are promising routes to potential transfer points; other routes are retained because they are tentatively the best routes to a city as well as because they are promising routes to potential transfer points.

As described above, the solution is geared to an originating time rather than to a desired arrival time. Another approach would have the program work from a desired arrival time and find the latest possible departure time from each of the other cities to that city. After the departure times are found, the earliest arrival times to the destination city based on each of the computed departure times could be determined. However, the approach described below should be more desirable because it finds good routes (fastest for some departure time) for all times of day in an efficient manner. If good routes for all times of day are known, the desired route for a given arrival time can easily be selected. The best routes for all times of day can be computed by means of the approach that follows.

The best routing for a given departure time, say 11:30 p.m., could be determined. Then the

best routes for a new departure time, say 10:30 p.m., could be considered in the following manner: Using as first links only flights which depart between 10:30 and 11:30, new potentially useful multiple-link routes would be computed. In deciding whether or not to save a computed route, it must be compared with the best route found so far, which includes those routes based on the 11:30 departure time. The computations involved would be less time-consuming for this second departure time because a set of flights would be saved only if it is better than the retained 11:30 departure routes. A third time, say 9:30, could be selected and the process continued by selection of earlier times throughout the day. The final results will be the fastest routes from the origin city to all other cities for each departure time used. (Duplicate routes can be suppressed.) From this mass of data, the routing that best answers the question originally posed can be selected. In addition, data are available to answer many similar questions.

This solution involves one minor problem. This can best be described by an example. Suppose we wish to arrive at a given city by midnight. If there are two direct flights, one which leaves at 10:50 p.m. and the other at 10:40 p.m. and both take an hour, the algorithm as described above would pick the flight with the earlier departure and arrival times, the flight leaving at 10:40. The 10:50 flight leaves later and would be the better flight according to the criteria described earlier in this paper. However, the selected flight is so similar to the best flight that this cannot be considered an important problem, especially since the interval between the departure times considered is subject to control.

An advantage of this approach is that a large amount of useful data is obtained in a systematic fashion.

LATEST SOLUTION

Routes to Be Stored

There is one property of the algorithm in the Appendix that warrants emphasis. This property can best be described using the network in figure one. Assume node A is the origin point. The shortest routes to node B and node C are the direct links of four and three units length, respectively. We then try to develop two-link routes to C, D, E, and F, using the link AB as the first link. The two-link route ABC is nine units in length, which is longer than the direct link. Although we compute the link ABC, we do not retain it. The links ABD and ABE, on the other hand, are retained. We then try using as a first link AC. The route ACB is computed but rejected. The route ACE (length 6) is computed and retained in place of the route ABE (length 8). In going through these operations we systematically consider all possible ways of extending each M link route to $M + 1$ link routes. In developing three-link routes, the route ABD is extended to make the route $ABDF$ and the route ACE is extended to make the route $ACEF$, which is longer than the stored route $ABDF$ and is therefore ignored. The route $ABDF$ is extended to the route $ABDFE$ which is longer than the route ACE, and therefore is not retained. In finding the route $ABDF$, the rule of using an M link route to find an $M + 1$ link route still held. There was no need to use any other procedure to find this route, such as extending the route AB by two links at one time.

Listing Procedure

In adapting the algorithm to find optimum transfer airports and flights, it is sometimes necessary to retain several routings from the origin airport to a given transfer airport. If time is the only criterion for the selection of an optimum route, the only criterion for retaining non-optimum routes to potential transfer points is also time, where the amount of time is a function of the required minimum transfer times at that city. We need consider only the arrival times at that one point along the route; we need not worry about transfer problems at previous cities along the route nor at cities yet to be added to the route.

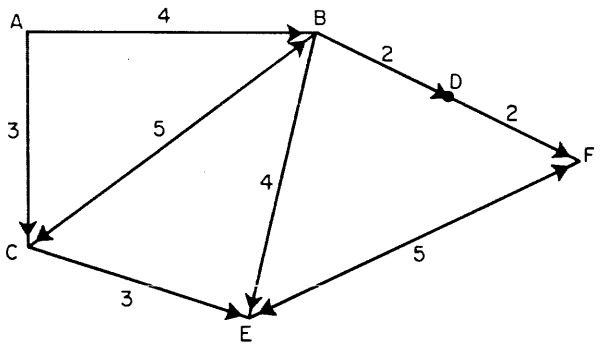
As indicated above, it is sometimes necessary to save more than one set of flights between the origin city and another city, simultaneously. To save space in the computer memory, the retained flights are stored in a list format. There is a limit on the space reserved for this list. However, there is no limit on the number of routes that can be stored between the origin city and a potential transfer point so long as there is space remaining in the list to store information regarding these routes.

Additional Criteria

The introduction of the list procedure permits additional route-selection criteria to be introduced. In introducing additional criteria, it is necessary to be very specific. Additional criteria that have been introduced and programmed are those of specific interest to the Post Office Department in routing air mail.

The Post Office Department pays for air transportation according to the following rules.

1. If only one airline is involved, the Post Office pays a loading charge based on the size of the airport plus a transportation cost based on the "short line distance" between the origin and destination airport. The short line distances are the shortest distances between the two airports involved, using a single carrier. For example, suppose we wish to get from airport A in Figure 1 to airport E and the routing we wish to consider is from A to B to E on airline 1. If airline 2 has planes between airport A and C and also has planes between C and E, airline one will be paid for only one loading charge and only six units of distance (the distance of the ACE route) rather than the eight units of distance that the mail was transported.



2. If more than one airline is involved, the short line distance is paid each airline for any continuous portion of the route handled by that airline. An additional loading cost is added each time the mail is transferred from one airline to another, based on the size of the airport at which the transfer is made.

It should be noted that when these rules apply, a straightforward application of the algorithm cannot answer questions regarding cost. However, as actual sets of flights are selected, costs can be computed. Whenever the fastest route is not the cheapest, the cheapest route found can also be saved. Hence, the criterion of cost, as well as that of speed, can be taken into account.

The introduction of the criterion of cost with the costing rules described above creates some problems. If we consider the problem of getting from A to B by way of T, the following could happen. Let us say that the best flight from A to T is on airline one, while airline two provides a flight which arrives much later. However, airline two provides the only service between T and B; therefore, the more time-consuming connections from A to T and B on airline two are cheaper, because there is no inter-line transfer cost. In other words, many poor flights and sets of flights must be retained and tested if it is required that the cheapest routing be found. This would greatly increase the computer time necessary. In order to make solutions practical, we have programmed the computer to select the cheapest set of flights that arrives at a transfer point less than X minutes after the fastest, where X is a variable set equal to, say, 120 minutes. Each of these selected sets is tested as the first m links of a route having $m+1$ links.

The problem of finding alternate routes when the fastest route does not operate every day is straightforward. Each computed routing can be tested to see if it is the fastest for any day of the week and if it is the fastest, it is retained.

The Program

A computer program has been written and debugged using the techniques described in this section which does the following:

1. It finds the fastest route from an origin city or airport to all other cities or airports.
2. It finds "cheapest" routes, using the rules described above.
3. It finds alternate routes when the fastest route does not operate every day.
4. It finds routes for all times of the day, using the procedure of finding the best flights for each of 24 different desirable departure times throughout the day, as described in section 3.
5. It can handle 50 cities and 2,000 non-stop flight segments.

The program was written in FORTRAN, with FAP function subprograms used to pack and unpack data for the IBM 7090. It takes about one second to compute the "best" routes from one airport to the other nine airports in a ten airport network with 200 trip segments for a specified earliest departure time. This computation time does not include "set-up" time nor the time required to enter the data.

APPLICATIONS

The procedures described in this paper have many potential areas of application. Two such applications related to the Post Office problems will suffice as examples.

The Post Office Department prepares lists of multiple-link routes for the routing of air-mail. It is anticipated that the procedures described in this paper will be used instead of a hand operation to develop these routes.

The Post Office Department schedules many mail trucks to supplement service provided by common carriers. The techniques described in this paper can be used to evaluate a proposed revision of schedules.

There are, of course, many other areas of potential use. In most of these cases, a corresponding hand operation is now being used. It is anticipated that computer procedures will be more efficient and more economical in many situations.

APPENDIX

A Computational Algorithm for Obtaining the Shortest Path From One Point to Every Other Point in a Network

Given a network of points $p_1, p_2, \ldots, p_m$ and lines between them, construct a distance matrix A, with elements a_{ij} representing the length of the line between points p_i and p_j. If no line exists between the points, let $a_{ij} = \infty$.

The algorithm also applies to the situation where the lines are directional. The value of a_{ij} would be the length of the line going from p_i to p_j. It would not be necessary that $a_{ij} = a_{ji}$.

Let e_i contain the ordered sequence of points of the shortest path found so far from p_1 to p_i.

Let b_i be the length of the shortest path found so far from 1 to i. The original values will be the direct distances, i.e., the first row of matrix A.

Let d_i indicate if the path e_i has been used in an attempt to create improved paths to other points. If $d_i = O$, it means it has been used, otherwise $d_i = 1$.

Let $f = 1$ if any d_i has been set equal to 1 since the last test of f, otherwise let $f = O$.

Steps

(1)	Set	$b_i = a_{1i}$	$i = 2, 3, \ldots, m$
	Set	$d_i = 1$	$i = 2, 3, \ldots, m$
	Set	$e_i = 1, i;$	$i = 2, 3, \ldots, m$
(2)	Set	$i = 2$	
	Set	$f = O$	
(3)	If	$d_i = 1$, go to step 7	
		$d_i = O$, go to step 4	
(4)	Set	$i = i + 1$	
(5)	If	$i \leq m$, go to step 3	
		$i > m$, go to step 6	
(6)	If	$f = O$, algorithm is finished	
		$f = 1$, go to step 2	
(7)	Set	$j = 2$	
(8)	Compute	$c = b_i + a_{ij}$	
(9)	If	$c \geq b_j$, go to step 11	
		$c < b_j$, go to step 10	
(10)	Set	$d_j = 1$	
	Set	$e_j = e_i, j$	
	Set	$b_j = c$	
	Set	$f = 1$	
(11)	Set	$j = j + 1$	
(12)	If	$j \leq m$, go to step 8	
		$j > m$, go to step 13	
(13)	Set	$d_i = O$	
		Go to step 4	

When the algorithm is finished, the contents of e_i, $i = 2, 3, \ldots, m$, will be the points through which a shortest path (more than one may exist) from p_1 to p_i passes. The values of b_i, $i = 2, 3, \ldots, m$, will be the length of the shortest paths.

It should be noted that the values of a_{ij} are used only in step 8. At that time, trial value of b_i is known and a_{ij} can be a function of that value of b_i. If b_i is the time required to get to i and a_{ij} is the time from the arrival at i to the arrival at j, then a_{ij} can be determined from published schedules using b_i in determining the earliest possible departure time at i.

REFERENCES

1. BELLMAN, RICHARD, "On a Routing Problem." *Quarterly of Applied Mathematics*, vol. 16, p. 87-90. 1958.
2. BOCK, F., KANTNER, H., and HAYNES, J., *An Algorithm (The r-th Best Path Algorithm) for Finding and Ranking Paths*

Through a Network. Chicago, Armour Research Foundation of Illinois Institute of Technology, Technical Paper Number 19. 1957.

3. DANTZIG, G. B., "On the Shortest Route Through a Network." *Management Science,* vol. 6, p. 187-190. 1960.
4. HOFFMAN, W., and PAVLEY, R., "A Method for the Solution of the Nth Best Path Problem." *Journal of the Association for Computing Machinery,* vol. 6, p. 506-514. 1959.
5. MINTY, GEORGE J., "A Variant on the Shortest-Route Problem." *Operations Research,* vol. 6, p. 882-883. 1958.
6. *Official Airline Guide,* Quick Reference Edition, vol. 6, No. 21. Chicago, Reuben H. Donnelley. 1962.
7. POLLACK, M., and WIEKENSON, W., "Solutions of the Shortest-Route Problem—A Review." *Operations Research,* vol. 8, p. 224-240. 1960.
8. SHIMBEL, A., "Structure in Communication Nets." *Proceedings of the Symposium on Information Networks,* New York, Brooklyn Polytechnic Institute. 1954.

EQUITABLE DISTRIBUTION

John A. Gosden
AUERBACH Corporation
1634 Arch Street
Philadelphia 3, Pa.

THE PROBLEM OF DISTRIBUTION

The problem of distributing available resources occurs in a great variety of networks, each with peculiarities of its own. Coal from mines has to be distributed to central dumps and to small yards. Ice cream must be distributed only to refrigerated stores and has a limited useful life. A buyer for a chain of stores with their own stock rooms but no central warehouse must indicate where his purchases are to be delivered. A farmer must distribute his labor according to the state of the crops and the weather, and many other distribution problems exist in modern complex commercial life.

The problem of distribution is that the sum of the requests or needs of consumers does not equal the available supply. This is due in part to failures to meet production or buying plans, changes in consumer demand and other uncontrolled variables, but also may be deliberate policy to maintain level production rates, to buy while raw material prices are favorable or to build up stock in expectation of peak demand periods such as Christmas or the Summer. The standard solution to the mis-matching of supply and demand is the creation of buffer stocks.

In practice, there may be one or many buffer stocks in a network. In retail selling, there may be 5 levels—display stock, counter stock, shop stock, area depot stock and factory warehouse stocks—apart from stocks in the pipelines between them. For all these, it is possible to establish optimum restocking periods and optimum restocking levels. In ideal conditions, the total available for distribution would equal the sum of the amounts by which each store was under-stocked and allocations would be made to bring each to its optimum level at each restocking period.

To illustrate the principle of equitable distribution, this paper considers an example consisting of a network in which there is one production site and several retail outlets. The whole network operates on a weekly cycle and the production in any one week is available at the end of the week to replenish the retail outlets. Each outlet has its own stock room but the production site carries no stock from week to week. Each week the forecasters estimate the weekly sales for each of the outlets. For example, suppose that it has already been established that at the beginning of each week the optimum restocking level of each of the outlets is ten days' estimated sales. The production plan, therefore, for any one week consists of the sum of the estimated sales of each outlet plus any under-stock and minus any over-stocks held in the outlets at the beginning of the week. If all goes well, production meets its targets, the sales volume is exactly that estimated and then the production is exactly sufficient to restock the outlets to their optimum levels at the end of the week.

However, at the end of the week, the situation that arises is not identical with the estimated situation. Three factors have had their

effects. First, production may be above or below that planned. Second, the actual sales may differ from those estimated. Third, the optimum restocking quantities may alter because estimates of future sales are revised. In such a situation, either the requirements of the outlets exceed the production volume and some form of under-supply must be imposed or production volume is in excess of the requirements of the outlets and some form of over-supply must be imposed.

There are two important restrictions which must also be taken into consideration. The first restriction is that no allocation shall be made that causes the stock level in an outlet to exceed the capacity of its stock room. The second restriction (which does not occur in all situations) is that no returns of stock from outlets or transfers among outlets are allowed. This latter restriction is a frequent feature of distribution networks.

Numerical Example I

Consider a system of one production site and four retail outlets in which the four outlets have storage capacities of 64, 60, 126 and 160 items, respectively. At the start of week one, the daily sales for each are estimated as 4, 4, 7 and 10 items, respectively. It is considered that the optimum restocking level at the start of each week is 10 days' estimated sales and they are so stocked.

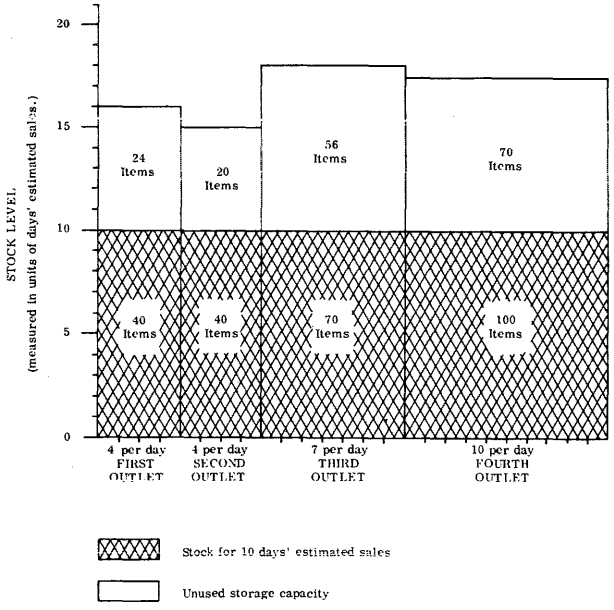


Diagram 1. Stock Position At Beginning Of Week 1.

Diagram 1 represents this stock position. On the horizontal axis, each outlet has a base representing one day's estimated sales and on the vertical axis, the height represents stock as a number of days' estimated sales. In this diagram, area is a measure of items. There are two areas shown for each outlet—occupied storage space and spare storage space. The storage capacities range between 15 and 18 days' sales. In this situation the production volume calculated for week one is 150 items, which is the total estimated week's sales with no corrections required for over- and understocks.

Suppose that during week one the sales are, in fact, 24, 35, 46 and 45 items, respectively. As a result of these sales, the forecasters make two changes, in which the estimated daily sales for outlets 2 and 3 are altered to 5 and 6 items. The position at the end of week one is then shown in Diagram 2. These changes have had

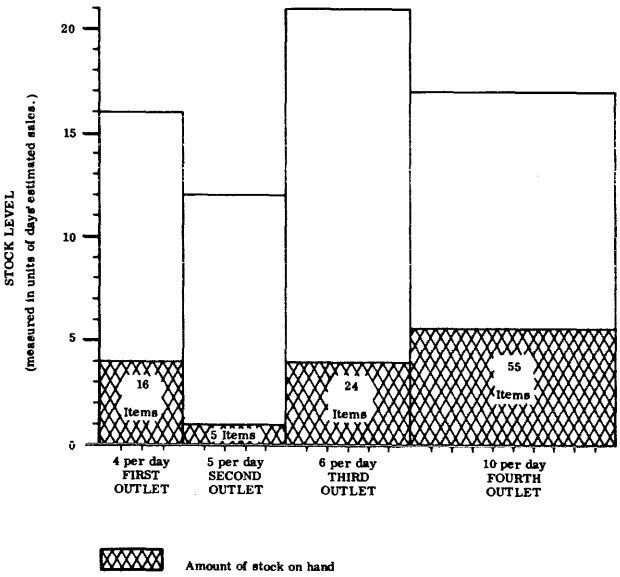


Diagram 2. Stock Position At End Of Week 1.

the important effect of reducing the capacity of the store at the second outlet from 15 to 12 days' sales, although it can still hold 60 items. The optimum stock levels of the outlets are now 40, 50, 60 and 100 items, and their closing stocks are 16, 5, 24 and 55 items; therefore, the current demands of the outlets to restock to the optimum level are 24, 45, 36 and 45 items, making a total demand of 150. Now suppose production was held up and there were only

100 items to distribute. If each demand is rationed by the same factor, and allocations of two-thirds of demand are made, the new position would be as shown in Diagram 3. This

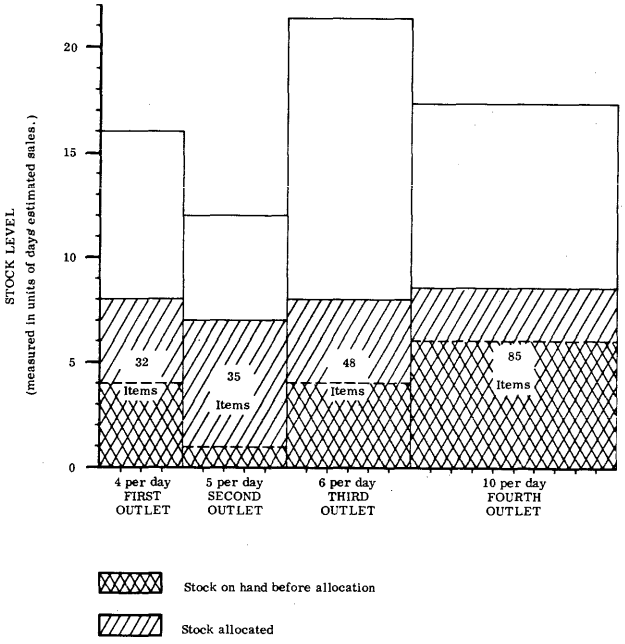


Diagram 3. Stock Position If All Demands Are Rationed Pro-rata.

would mean that the outlets had stocks of 8, 7, 8 and 8.5 days' sales, respectively. This system causes those that have been selling well to have the poorer stocks even when the forecasts have been corrected. On the other hand, the principle of equitable distribution would allot 16, 35, 24 and 25 items to the outlets, which would mean that each had a stock of 8 days' expected sales as shown in Diagram 4.

Simple Model

The Diagrams are constructed in such a way that after equitable distribution, the stocks are represented by rectangles of equal height, and a simple model can be used to illustrate the principle of equitable distribution. Let the Diagrams represent a vertical cross-section of a rectangular tank in which the floor is divided into rectangular strips, perpendicular to the cross-section, one for each outlet. The widths of the strips are proportional to the rate of sales of each outlet. Solid blocks are inserted into the tank to represent the occupied storage space for each outlet. Each block fits exactly onto the strip corresponding to its outlet, and

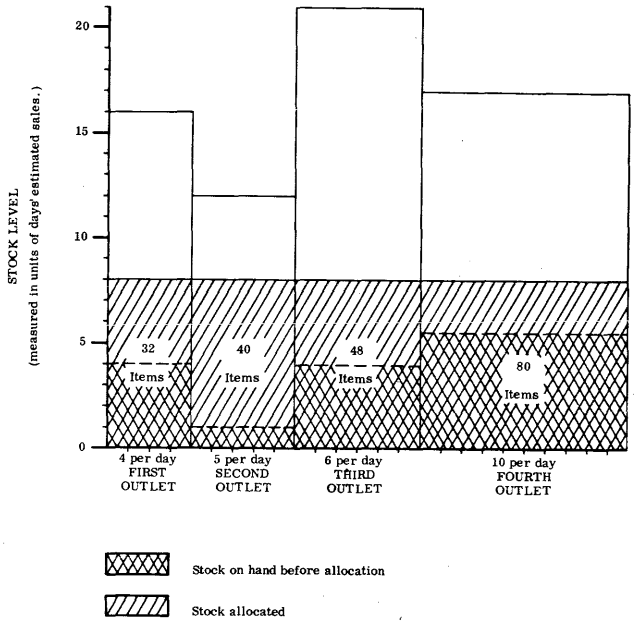


Diagram 4. Stock Position If Distributed Equitably.

its height, therefore, measures the stock as a number of days' expected sales. The tank is completed by a stepped roof whose height above the floor of the tank represents the limit of each outlet's storage capacity.

Now let the week's production be represented by a suitable volume of liquid which is poured into the tank. The liquid will find its own level, as shown in Diagram 4 for the case when the volume is 100. Diagrams 5 and 6 illustrate

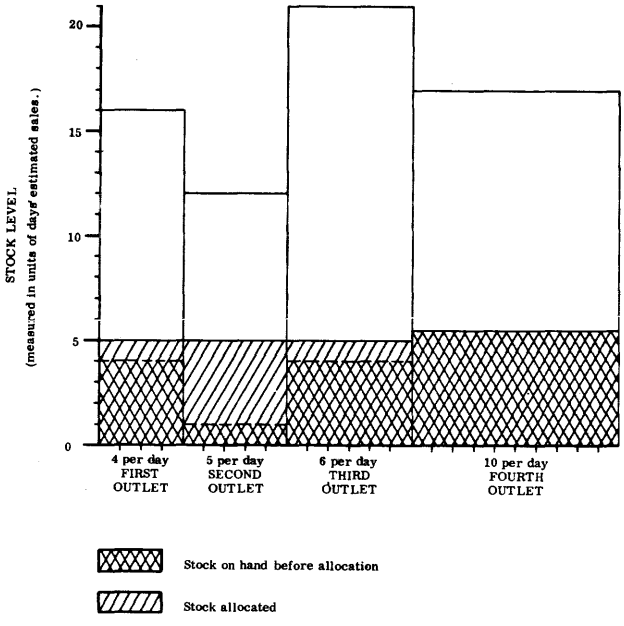


Diagram 5. Equitable Allocation of 30 Units.

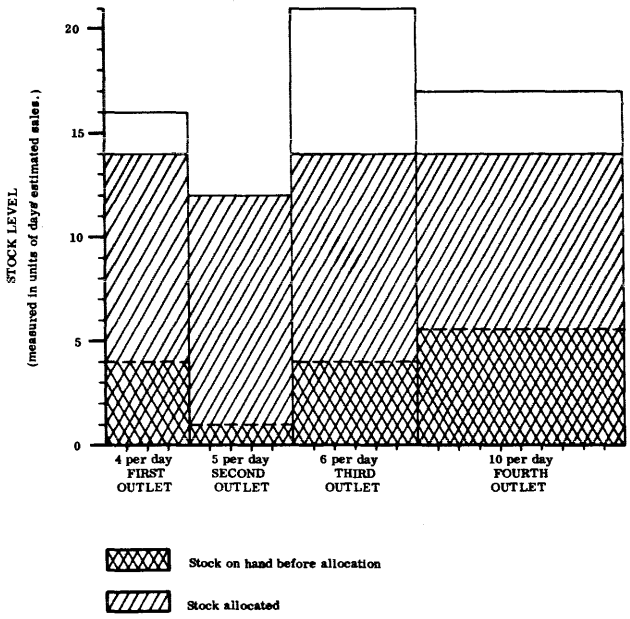


Diagram 6. Equitable Allocation of 240 Units.

alternative cases for production volumes of 30 and 240, respectively. Diagram 5 shows how the fourth outlet is ignored when its level of stock exceeds the equitable level, and Diagram 6 shows how the roof of the tank limits the allocation made to the second outlet.

Computation Procedure

The computation procedure consists of two parts: the first determines the level of equitable distribution; the second computes the appropriate volume of production to be allocated to each outlet.

Suppose there are N outlets. Let the estimated daily sales of outlet n be S_n items, the stock capacity be C_n days' sales, i.e., C_nS_n items, and the actual stocks be A_n days' sales, i.e., A_nS_n items. Now C_n and A_n denote the discontinuities in the system and let them be ordered into a series B_i where $i = 0, 1, 2, 3, \ldots, i_m$,

where $B_i \leq B_i + 1$
and $B_0 = 0$.

Now calculate X_{in}, which is the volume required to bring up the stock of outlet n from B_{i-1} to as close to B_i as the capacity C_n allows.

If $B_{i-1} \geq C_n$, $X_{in} = 0$
If $B_i \leq A_n$ $X_{in} = 0$
Otherwise, $X_{in} = [\min(C_n, B_i) - \max(A_n, B_{i-1})]S_n$.

Compute F_i where

$$F_i = \sum_{i=1}^{i} \sum_{n=1}^{N} X_{in}.$$

Then F_i is the volume required to bring up the stocks of all outlets to as close to B_i as the capacities allow.

Let P be the volume of production to be allocated. Then find I such that

$$F_I \leq P \leq F_{I+1}.$$

Then the level L of equitable distribution is defined by

$$L = B_I + Q(B_{I+1} - B_I)$$

where

$$Q = (P - F_I)/(F_{I+1} - F_I).$$

The allocation D_n for outlet n is defined by

$$D_n = \sum_{i=1}^{I} X_{in} + QX_{(I+1)n}.$$

These allocations have the property that their sum is exactly equal to the volume of production, because

$$\sum_{n=1}^{N} D_n = F_I + Q \sum_{n=1}^{N} X_{(I+1)n}$$
$$= F_I + (F_{I+1} - F_I)(P - F_1)/(F_{I+1} - F_I)$$
$$= P.$$

Numerical Example II

Table 1 shows the values of S_n, C_n, A_n, for the four outlets. The values of B_i are shown in Table 2, and they correspond to the discontinuities marked in Diagram 2. Table 3 shows the values X_{in} computed for the situation illustrated in Diagram 3. The column totals are cumulatively cross-cast at the foot to give the values F_i. Three cases are illustrated in which P (the volume of production) is 100, 30 and 240, respectively.

Case 1, when $P = 100$:
$F_4 = 37.5 \leqslant 100 \leqslant 200 = F_5$
$Q = (100 - 37.5)/(200 - 37.5) = 625/1625$
$= 5/13$
$L = 5.5 + 5(6.5)/13$
$= 8$ days' sales.

Case 2, when $P = 30$:
$F_2 = 15 \leqslant 30 \leqslant 37.5 = F_3$
$Q = (30 - 15)/(37.5 - 15) = 15/22.5 = 2/3$
$L = 4 + 1.5(2/3)$
$= 5$ days' sales.

Case 3, when $P = 240$:

$F_5 = 200 \leqslant 240 \leqslant 280 = F_6$

$Q = (240 - 200)/(280 - 200) = 0.5$

$L = 12 + 0.5(4)$

$= 14$ days' sales.

Quantity	Code	Outlet 1	Outlet 2	Outlet 3	Outlet 4
Outlet Number	j	1	2	3	4
Estimated Daily Sales	S_n	4	5	6	10
Stock Capacity In "Days Sales"	C_n	16	12	21	17
Actual Stock In "Days Sales"	A_n	4	1	4	5.5

i	B_i In Days	Discontinuity
0	0	Empty stockrooms
1	1	Current level outlet 2
2,3	4	Current levels outlets 1 and 3
4	5.5	Current level outlet 4
5	12	Capacity level outlet 2
6	16	Capacity level outlet 1
7	17	Capacity level outlet 4
8	21	Capacity level outlet 3

i	B_i (Days)	Values of X_{in}				ΣX_i all outlets	F_i
		outlet 1	outlet 2	outlet 3	outlet 4		
1	1	0	0	0	0	0.0	0.0
2	4	0	15	0	0	15.0	15.0
3	4	0	0	0	0	22.5	37.5
4	5.5	6	7.5	9	0	0.0	37.5
5	12	26	32.5	39	65	162.5	200.0
6	16	16	0	24	40	80.0	280.0
7	17	0	0	6	10	16.0	296.0
8	21	0	0	24	0	24.0	320.0

These cases are illustrated graphically in Diagram 8. The graph is formed by joining the pairs of points (B_i, F_i) computed from Table 2. By taking the value of L corresponding to the value P on the F axis, the same answers are obtained as when using the algebraic method given above.

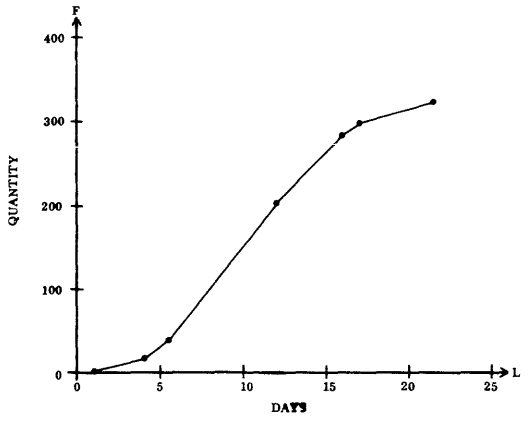


Diagram 8. Quantity F Required To Raise Minimum Stock Level to L Days' Estimated Sales.

Practical Cases

In practice, the procedure detailed above is not suitable for general computation. The procedure suffers from the disadvantage that the determination of the series B_i is not straightforward. In addition, it produces many more B_i values than are usually necessary for the precision required, and which increases the computing unnecessarily. With 50 outlets in the example above, there would be nearly 100 values of B_i. If, on the other hand, arbitrary choices of B_i are made, the effect is to produce a graph only slightly different from Diagram 8. Table 4 shows a possible set of B_i, and the

i	B_i (Days)	Values of X_{in}				X_i	F_i
		outlet 1	outlet 2	outlet 3	outlet 4		
1	3	0	10	0	0	10	10
2	6	8	15	12	5	40	50
3	9	12	15	18	30	75	125
4	12	12	15	18	30	75	200
5	15	12	0	18	30	60	260
6	18	4	0	18	20	42	302
7	21	0	0	18	0	18	320

F_i that result from it. All these points are on the graph shown in Diagram 8, but their connection lines sometimes cut across concavities of the more accurate graph. This is shown most significantly for small values of L. Diagram 9 shows the precise graph as a solid line and the approximation as a broken line.

Re-working Case 2, using Table 4, gives

$Q = (30 - 10)/(50 - 10) = \frac{1}{2}$

$L = 3 + 3/2$

$= 4.5$ days' sales,

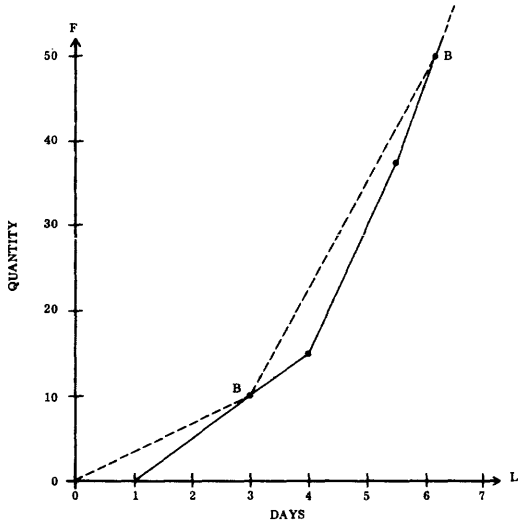


Diagram 9. Enlargement of Diagram 8 Showing Approximation Caused By Arbitrary Selection of Points B.

but L in this context is only an approximation. It is more interesting to note that the allocations that would be made are

4, 17.5, 6, and 2.5

which are very close to the preferred amounts

4, 20, 6, and 0

and would bring the stock levels up to

5, 4.5, 5, and 5.25

days' sales, with a maximum error of 0.5 days' sales. This is much better than rationing the available 30 by the fraction 30/150, which would allocate

4.8, 9.5, 7.2 and 9,

giving stock levels of

6.4, 2.9, 5.2 and 6.4

days' sales respectively, in which case the second outlet is badly understocked. By setting the B_i at one-day intervals, the approximation error would be negligible.

Back-Orders

If a situation occurs where stock runs out and back-ordering is involved, then an area should be added below the horizontal axis representing the back-orders. Diagram 7 shows how 100 items would be allocated if at the end of week 1, the first outlet had 8 left, the fourth outlet had 10 left, and the others had sold out, while the third outlet had taken orders for 18. Each would have its stock brought up to a level of 4 days' sales.

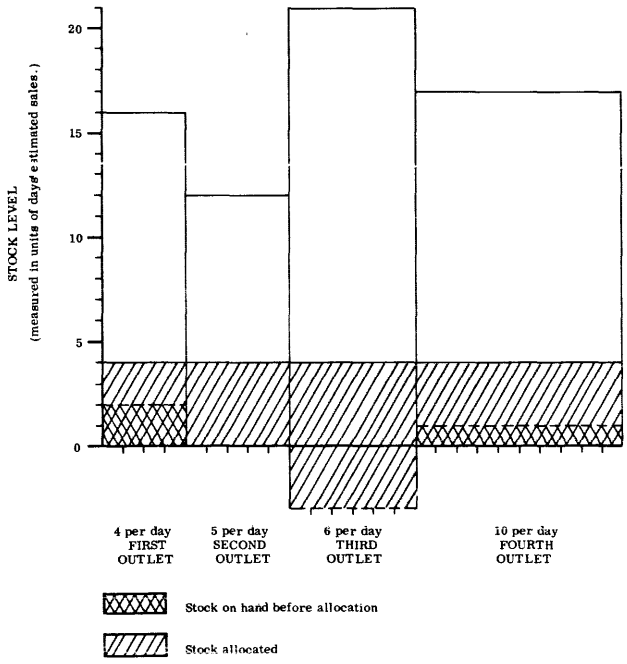


Diagram 7. Allocation Allowing For Back Orders.

Negative Allocations

In some networks, transfers between outlets are permitted. These may be direct transfers, or indirect transfers by means of the distribution point at the production site. There is always a stock limit set below which items may not be transferred. In the extreme, this is the zero stock level, but is more likely to be some emergency level of stock. Let the emergency level be one of the values of B_i, say B_E. Then the procedure is the same as before except that it is preceded by reducing the stock of each outlet to the level B_E and increasing P by the sum of these reductions.

Indefinite Capacities

Where an outlet deals only in one kind of item, or where a bin system or special storage is required (say for gasoline or frozen foods), there are definite values that can be set for the C_n, but in many cases the value of C_n varies as other items held in the same store are under- or over-stocked. In this situation, let there be J items considered and all the symbols used before now have an extra suffix j (j=1, 2, 3 . . . J). Now values C_{nj} are allotted which must conform to the restriction that when V_j is the volume of an item j, then,

$$\sum_{j=1}^{J} C_{nj} V_j$$

is less than the total capacity of outlet n. The values C_{nj} should be set at a level above which any extra stocks would not materially decrease the chances of running out of stock.

Having allotted C_{nj} for all j for some outlet n, let the storage space remaining, called the reserve storage, be R_n measured in some unit of volume. For convenience, we consider the reserve storage in terms of capacity to hold some item j whose volume is V_j. Then $R = V_n E_{nj}$ for any j and E_{nj} is the reserve capacity of outlet n in units of item j.

If P_j is greater than the total allocations necessary to bring up the stock of each outlet to its C_{nj}, then the reserve capacities must be used, and the remaining part of P_j is called reserve

stock. The reserve stock must be compared with the total reserve storage space which is

$$\sum_{n=1}^{N} E_{nj} V_j \quad \text{for any } j.$$

If the volume of reserve stocks for all items together exceeds reserve storage space, the values of P_j must be reduced by some system until it is the case. Let H_j be the fraction of total reserve storage required by the reserve stock of item j. Then, where reserve stock exists, each outlet is filled to capacity C_{nj} and then allocated a sufficient amount to bring its reserve holding to $H_j E_{nj}$.

Computer Considerations

There are five major factors to be considered:

1. Volume of computation.
2. Complexity of computation.
3. Scans of the data.
4. Random or serial access to data.
5. Accuracy and rounding errors.

The volume of computation depends upon the number of outlets and the number of points B_i. The more B_i points there are, the greater the precision of the result in general. For any given degree of precision, an arrangement of B_i in exponential steps minimizes the number of B_i.

The procedure requires scans through the data to:

1. Establish the values of B_i. This scan can be saved if preset values are used.
2. Compute the values of F_i, and then determine L.
3. Compute the individual allocations D_n.

Thus there are either 2 or 3 scans, depending upon the method of establishing the values of B_i.

If the data for the outlets can be held in a random access store, the scan to determine L can be made by computing each F_i in turn until the allocation total P is reached. A more sophisticated procedure is to estimate I and arrange a searching procedure. More frequently, the data may have to be held in a serial access store. In this case, the procedure is to calculate all the X_{in} for each n in turn and accumulate

$$\sum_{n=1}^{N} X_{in} \quad \text{for each } i.$$

Then at the end of the scan, the values F_i can be formed.

Although the computation of F_i may be made in basic items, allocations D_n may have to be rounded to some multiple of a delivery unit. When this unit is not small compared with the smaller values of D_n, the round-off procedure needs care. Any errors introduced by rounding are additional to those arising from the choice of the set of B_i. Rounding cannot be carried out indiscriminately or the sum of D_n may not equal P. If independent rounding procedures are carried out for each outlet, it is wise to group the outlets with larger S_n at the end of the scan where cumulative round-off errors can be absorbed with least inequity. If such regrouping is not desirable, the technique of carrying the round-off quantity into the computation of the next outlet can be used. For example, suppose D_n be rounded to D'_n and $D'_n - D_n = d_n$ then D'_{n+1} is computed by rounding off $(D_{n+1} - d_n)$. This process minimizes the cumulative round-off error at any time, and each D'_n may have an error of up to one delivery unit.

CONCLUSION

There are five major advantages that can be cited for the use of the principle of equitable distribution that has been discussed in this paper.

First, the basis of the principle is simple to understand; therefore, it is easy to explain to all employees concerned with distribution.

Second, the principle works just as easily in times of crises as in easier times. During potential crises—heat-waves, blizzards, or failures in production or raw materials—supplies are rationed with little fuss. During inventory build-up to minimize a future potential crisis, the surplus is shared simply and effectively.

Third, the principle is easy to mechanize and fits well into automatic data processing and simpler systems.

Fourth, the principle can be extended to cover more complex situations where several levels of inventory or sources of supply are involved.

Fifth, it can be used effectively with other mathematical tools. Starting from past sales records, exponential weighting techniques[1, 2, 3]

can be used to forecast sales. From sales patterns and production costings, mathematical analysis[4, 5] can be used to determine optimum inventory levels and re-ordering rules. From current stocks and sales forecasts, production plans can be made; then, when production occurs, equitable distribution can be made.

ACKNOWLEDGMENTS

The author wishes to thank Mr. T. R. Thompson of LEO Computers, Ltd., and the staff of AUERBACH Corporation for their help and encouragement in the preparation of this paper.

REFERENCES

1. BROWN, ROBERT G. (1956) Exponential Smoothing for Predicting Demand. Cambridge, Mass., Arthur D. Little, Inc.
2. HOLT, CHARLES C. (1957) Forecasting Seasonals and Trends by Exponentially Weighted Averages. Pittsburgh, Pa., Carnegie Institute of Technology.
3. MUIR, ANDREW (1958) Automatic Sales Forecasting. The Computer Journal, Vol. 1, p. 113.
4. WHITIN, T. M., YOUNG, J. W. T. (1955) A Method for Calculating Optimal Inventory Levels and Delivery Time Naval Research Logistics Quarterly, Vol. 2, No. 3, p. 157.
5. ARROW, K. J., HARRIS, I., MARSCHAK, J. (1951) Optimal Inventory Policy. Econometrica, Vol. 19, p. 250.

RAMPS — A Technique for Resource Allocation and Multi-Project Scheduling

Jack Moshman, Jacob Johnson, and Madalyn Larsen
C-E-I-R, INC.
1200 Jefferson Davis Highway
Arlington 2, Virginia

INTRODUCTION

Some Recent Developments

Work planning, a never-ending management responsibility, has been aided tremendously in recent years by the development of a new technique commonly referred to as networking or arrow-diagramming. Today, the network is widely accepted by business, scientific, and governmental organizations as a worthy replacement for the Gantt chart and other less flexible and less meaningful methods of planning work.[1-3, 9, 12]

PERT,[4, 10] Critical Path Method (CPM),[7, 8] and many other similar systems[5, 6] use estimates of the time required to complete each activity as the basis for determining the work schedule. The scheduling system sequences all the activities in the network and calculates the earliest and latest completion dates for each activity. These dates are then woven together to form a schedule for the total project. In some instances the scheduling function is automated, that is, an electronic computer is used to perform the calculations; in others the schedule is determined manually and monitored with the aid of a computer.*

RAMPS, a system for *R*esource *A*llocation and *M*ulti-*P*roject *S*cheduling, was recently developed by C-E-I-R, INC. and now is an operational IBM 7090 digital computer program. RAMPS retains many of the basic concepts of its predecessors; it uses the network for work planning and relies on a careful analysis of the needs of each individual activity, but it also has unique features not found in other systems. Two of these are readily apparent in its name—Resource Allocation and Multi-Project Scheduling.

Activity Resource Requirements

Major differences among many networking systems lie in the information provided for each activity and ultimately used in the work scheduling function. In most systems, this information includes estimates of the time needed to complete each activity and the total cost of the activity or a related group of activities. Where a schedule is produced, it attempts to reflect the most desirable time-cost relationships.

While RAMPS includes time and cost considerations in its work schedules, it also incorporates the *resource requirements* of each activity and the *availability of these resources* at the time the activity is to be scheduled—both extremely vital factors in any meaningful scheduling system.

Multi-Project Schedules

A unique feature of RAMPS is its ability to schedule *simultaneously* more than one work

* A comprehensive bibliography is to be found in Voress, *et al.*[11]

project. The projects to be scheduled may differ in size, type of work, importance and starting times. They are related only in their reliance on a common pool of resources.

RAMPS recognizes and responds to established priorities for the projects and competition among activities within all projects for limited quantities of available resources. The system also strives to meet established target completion dates by applying larger quantities of available resources to critical activities within all projects.

Competition for Available Resources

In addition to defining the work and resources required by the various activities, the RAMPS user provides the system with a knowledge of the quantity of each resource type that is available to all projects. Provision is also made for using overtime or additional units of a given resource at a premium cost.

There may be many activities in all projects competing for the same resources during the same work period. RAMPS weighs the needs of each activity individually and in relation to the other activities before deciding how the resources are to be allocated.

Management Controls

Under the many constraints imposed by completion deadlines, specified resource limits and project priorities, RAMPS must weigh many factors before deciding how best to schedule each project. Frequently, there are many possible routes that RAMPS could follow, each with a different effect on the schedules produced. There is, for example, the route that minimizes project completion time, but perhaps at an increased cost because of the use of overtime. Another route may assure a minimum of idle resources throughout the lives of the projects, and another might maximize the total number of activities being worked on during each scheduled work period. In instances such as these, RAMPS relies on control information provided by the RAMPS user to determine which course of action is most desirable. This ability on the part of management to influence and guide the scheduling function is one of the major features of the RAMPS system.

ESTABLISHING THE NETWORKS

General Description

The foundation of the RAMPS system is the network—a graphic display of a plan. The network portrays an orderly step-by-step series of actions which must be performed successfully in order to reach a specific, definable objective. Simply stated, a network is a work flow diagram.

Concurrency in the Network

In almost every real situation, there are many activities that can be carried on concurrently; others must be accomplished in a purely serial fashion. By planning to allow several related efforts to proceed simultaneously and converge at the proper event, the manager is able to reach his stated objective in a much shorter period of time. Since the network is a work plan, those activities which logically can be worked on in parallel should be shown in the network as concurrent activities. The concept of serial and concurrent activities is shown in Figure 1.

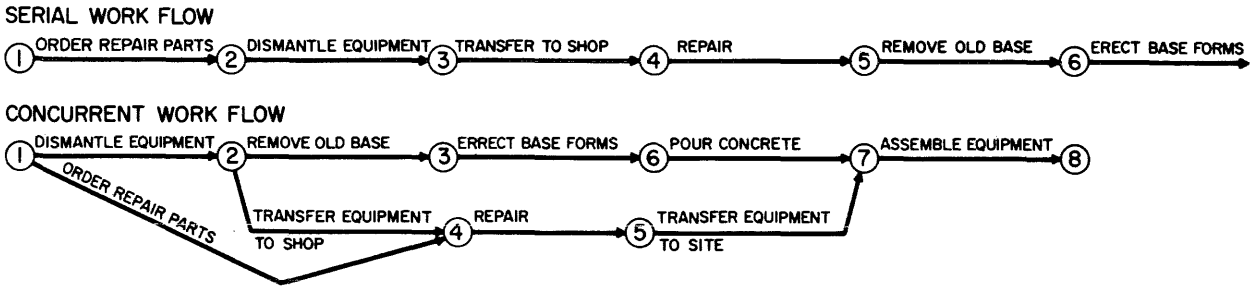


Figure 1. Concept of Concurrent Work Flow and Serial Work Flow.

It is obvious that a considerable amount of effort must be expended to determine which activities may be concurrent and which must proceed alone. But it is this effort at the planning level that saves time and money later when the actual work is done.

ESTIMATING TIME AND RESOURCE REQUIREMENTS

General Description

Although the importance of an accurate and well-planned network cannot be over-emphasized, there is perhaps no other function in the total application of RAMPS that is more important than obtaining accurate estimates of the time and resources required by *each activity*. Because RAMPS bases many of its scheduling and resource allocation decisions on this information, the validity of the schedules produced hinges on the thoroughness with which time and resource requirements are estimated. It is imperative that these estimates be made by those individuals who are most familiar with the work to be accomplished by each activity.

Determining Amount-of-Work

With the preliminary networks drawn and available as work guides, the next step in applying RAMPS can begin: determining the *amount-of-work* required by each activity in the networks.

Amount-of-work is derived from multiplying the number of unit time periods required to complete an activity under *normal working conditions* by the number of units of resource required per time period. The unit time period may be an hour, day, month, or any unit of time that defines the smallest period within which work will be scheduled and resources allocated.

Although it is not necessary to record amount-of-work in the network, it is frequently beneficial because it provides a ready visual display of the time and resource estimates for each activity as illustrated in Figure 2. The amount-of-work figures are enclosed in boxes below the activity lines. The units of resource required per time period are recorded beside the amount-of-work boxes. The estimated number of time periods needed to complete each activity can be quickly determined by dividing the units of resource figure into the amount-of-work.

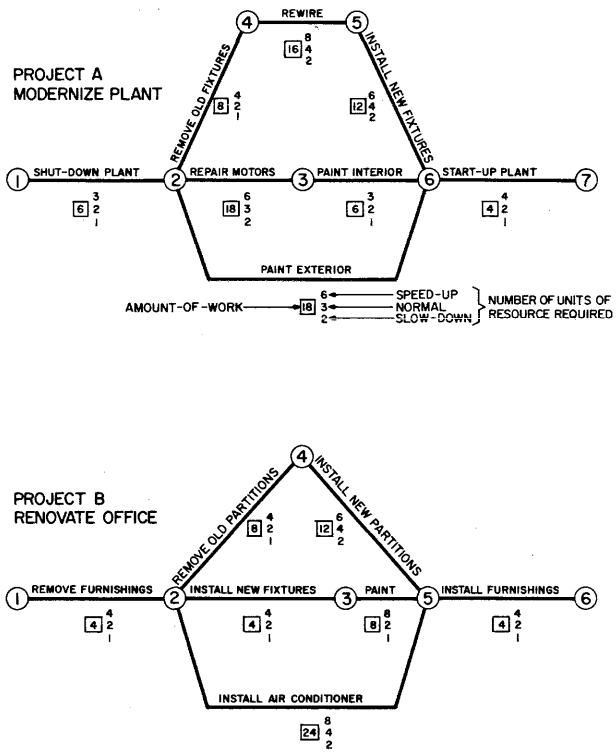


Figure 2. Amount-of-Work and Alternate Resource Utilization Rates in the Networks.

If we assume that under normal conditions activity 3-6 of Project A will require three days to be completed and will consume the work of two painters during each day, the amount of work for the activity would be six units.

The amount-of-work concept provides the system with unique flexibility in work scheduling and resource allocation. By including two additional resource utilization rates to the normal rate already established, this flexibility can be further increased.

Establishing Alternate Utilization Rates

Under normal conditions, activity 3-6 would require three time periods to be completed and would use two painters per time period. To provide for the possibility of doing the job faster or slower than normal, one can provide two other estimates. The first is a resource utilization rate under accelerated work conditions, *speed-up;* the second is a resource utilization rate under relaxed or extended work conditions, *slow-down.* The work efficiency at other than the normal rate is introduced to account for the absence of precise linear relationships.

Once the amount-of-work has been determined for an activity, it should be the guiding factor in determining desirable speed-up and slow-down utilization rates. It is necessary to "tailor" utilization rates and work efficiencies for a given resource to the individual activity to which the resource is to be applied. Each activity in the network and the resources it needs are considered as autonomous units for purposes of estimating amount-of-work, resource utilization rates, and work efficiency at the three utilization rates.

Scheduling and Allocating Flexibility

The three rates of resource utilization provide RAMPS with great flexibility in manipulating time and resources requirements of each activity to meet *resource availability levels*. The same flexibility extends from the activity level to the project level where the speed-up, normal, and slow-down rates allow the system to adjust work accomplishment rates to meet project completion deadlines.

As shown in Figure 3, Project A could be completed in as few as 9 time periods at the speed-up rate, or as many as 32 time periods at the slow-down rate. At the normal rate, the project could be completed in 16 time periods. Note that the use of each rate requires a different peak work force. The total work force required reaches peaks of 20 men during period 5 at speed-up, 10 men during period 8 at normal, and 6 men during period 15 at slow-down rates.

If all the needed resources were available, it is likely that RAMPS would schedule a project at the speed-up rate. However, in real situations, all the needed resources are rarely available at all times, especially when there are several projects involved.

Let us assume, therefore, that only 7 men are available for work on Project A. Under this restriction, we can examine the steps taken by RAMPS in developing a schedule for that project considered in isolation. We will also see how the three utilization rates are intermixed in the schedule produced.

Determining the Critical Path

One of the first uses RAMPS makes of the amount-of-work values is in determining the *critical path* within each project. The critical path is the longest path or sequence of activities, in terms of *total time required*, from the starting to the ending activity.

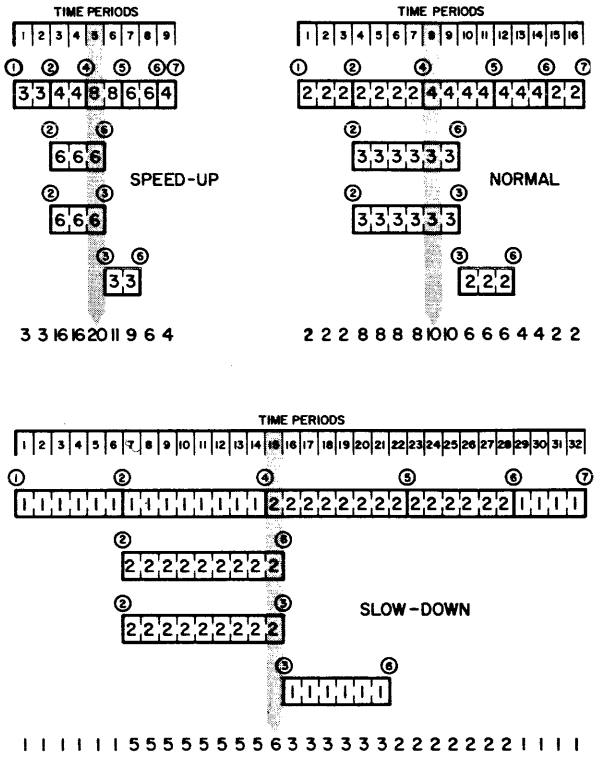


Figure 3. Possible Project Completion and Workforce Requirements at each Rate of Homogeneous Resource Utilization (Project A, Figure 2).

All activities on the critical path are *critical activities*. As shown in Figure 2, there are three possible work paths in Project A, the longest of which requires 16 time periods at normal rates and travels through events 1, 2, 4, 5, 6, and 7. This path is the critical path; any delay in the completion of a critical activity will cause an equal delay in completion of the project. If any or all of the activities *not* on the critical path are completed ahead of schedule, there could be no time gained in project completion. On the other hand, time gained along the critical path means time gained in project completion. Thus, the critical path calculation provides the following information:

1. The duration of the project if all activities on the critical path are scheduled at the normal resource utilization rate, and
2. The identity of those activities which are critical and therefore must receive *preference* when they are competing for a limited resource with activities that are *not* on the critical path.

Therefore, the next step in developing a schedule is determining when there will be competition between critical and non-critical activities for the 7 men that are available. This is done by establishing the earliest possible *start times* for the non-critical activities so that the *total resource requirements* for all activities *in each time period* can be determined. Figure 3 shows the earliest periods at which work on each activity in the project can begin. Note that beginning in period 4, the resources required at normal rates exceed the quantity available.

It can be seen that by using the slow-down utilization rates during periods 4 through 9, a schedule could be produced that stays within the limits of the available resources. However, this can be done only at the expense of extending the project completion time.

Since RAMPS strives to complete each project as quickly as possible, the use of slow-down rates on critical activities is essentially a last resort. Therefore, another alternative must be considered: delaying the start of the non-critical activities so that the resources they would otherwise consume can be diverted to the critical activities. This is called "floating" an activity.

Determining Activity Float

Activity float is the difference in time periods between the earliest time an activity *can* be completed and the time it *must* be completed without extending the project completion time. An activity float analysis for Project A is shown in Figure 4. Note that the critical activities have zero float—they cannot be delayed without delaying project completion.

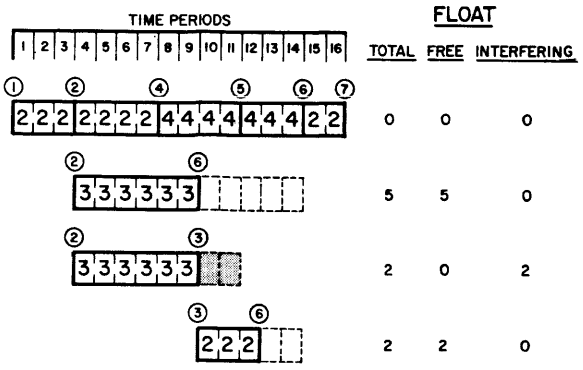


Figure 4. Activity Float Analysis.

The float for activity 2-6 is five time periods. The earliest time it can be completed is period 9; it must be completed during period 14 to preclude a delay in the start of activity 6-7. This kind of float is called *free float* because the activity can be delayed without interfering in any way with the float of other activities.

Conversely, activity 2-3 has two periods of *interfering float*. Since it must be completed before activity 3-6 can begin, 2-3 can be delayed one or two time periods, but only with an equal reduction in the float of activity 3-6. For this reason, the float of activity 2-3 is said to interfere with the float of an activity that is to be started later.

Producing a Schedule

With the combined power of the float analysis and the three rates of resource utilization, RAMPS is now equipped to produce an efficient schedule that meets the work requirements of each activity, minimizes project completion time, and stays within the limits of available resources. An idea of how this power is used can be gained from Figure 5 which shows the schedule produced for Project A.

By delaying the start of activity 2-6 and using the speed-up and slow-down rates where necessary, RAMPS has scheduled the project for completion in 15 time periods using a total work force of only 7 men. Note, too, that idle resources have been minimized where possible.

Although this small example serves to illustrate how RAMPS schedules work, a better idea of the scheduling power of RAMPS can be gained when one considers that this example takes into account only one project and only

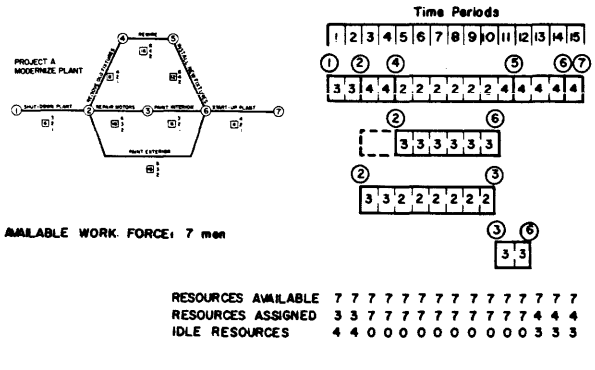


Figure 5. Derived Work Schedule for Project A Using Float and Combined Resource Utilization Rates.

one resource type: men. Looking again at the network for Project A, one sees that although men are required in each activity, they have different skills. There are carpenters, painters, electricians, and other skills involved. Therefore, in scheduling this project, RAMPS would not only have to consider total work force, but also the availability levels of each of the skills in each time period.

RESOURCE TEAMING

In our discussions to this point, we have included only the contingency of an activity requiring one resource type. There is a need, of course, for a means of applying several different resources to the same activity while still maintaining the amount-of-work concept. In the RAMPS system, this need is fulfilled by *resource teaming.*

When an activity requires several different resource types, they are combined to form a resource team for the activity. Each resource team is composed of a *lead resource* and one or more *trailing resources.*

The lead resource can be considered the resource in the team upon which the greatest overall demand is levied. It then is used to determine the amount-of-work for the activity and the slow-down and speed-up completion rates. The trailing resources are those "subordinate" resources that are needed to support the work to be done by the lead resource. Although trailing resources may actually perform work on the activity, they do not enter into the amount-of-work calculation. Amount-of-work for resource teams is derived solely from the lead resource date but all resources must be available before the activity can be scheduled.

DETERMINING RESOURCES AVAILABLE

Having covered the first two steps in the application of RAMPS:

1. Project planning and network preparation, and
2. Amount-of-work estimates and resource requirements for each activity in each project,

one must form the resource pools from which RAMPS will allocate the needed resources to the various activities and projects. The task is to determine:

1. How much of each resource type is available,
2. When and how long this quantity is available, and
3. The resource cost.

For each resource, these questions must be answered from two points of view. First, we must determine the answers under *normal* working conditions or at normal cost. Then we must establish the various means by which additional resource units could be provided if needed. The possibilities include overtime, subcontracting, and hiring or acquiring additional units.

These additional units are called *premium resources* and, as the name implies, their unit costs are usually higher than those of the normal resources.

Obviously, normal availability data is required by the RAMPS system; if work is to be scheduled, resources must be provided. Premium availability data is not required, but it is important in providing RAMPS with further flexibility in accomplishing the required work. Certainly the application of additional resources, derived by any means, will contribute to the completion of the required work in a shorter period of time, but at an additional cost. This data is particularly important when management is willing to incur greater expense to complete a project in a minimum amount of time. The quantity of any resource may be constant over the total scheduling period or may vary in level as a function of time.

Additional resource information needed includes the unit cost per time period, the availability of additional premium resources, and the cost of premium resources.

Although RAMPS strives to avoid allocating premium resources, it may use them under either of the following two conditions:

1. To meet a project completion deadline when no other means exists for accomplishing the required amount of work within the time specified.
2. To alleviate an otherwise untenable scheduling bottleneck caused by an acute shortage of one or more resources during one or more time periods.

SETTING MANAGEMENT CONTROLS AND OBJECTIVES

General Description

The next, and final, step before operating on the data accumulated so far is the setting of the objectives to be reached in the schedules that are produced. The use of the various controls determines project priority and provides answers to such questions as: "In scheduling all projects, should cost be minimized? Is time important? Should idle resources be minimized?"

Establishing Project Priorities

In perhaps every instance of multi-project scheduling, management can pinpoint those projects which, when completed, will be more beneficial to the company as a whole. Naturally, if these benefits are to be realized as early as possible, these projects must be given priority when all projects are competing simultaneously for a limited quantity of available resources.

Project priority can also be viewed from a slightly different angle: "How much is it going to cost if this project is not completed on time? How much can be saved or gained if it is completed ahead of time?" The answers to these questions are crucial in the many areas of business where work projects are negotiated under bonus-penalty agreements.

The relative importance of the various projects is injected into RAMPS decision-making by providing the system with three facts: *project start dates, desired project completion dates,* and *project delay penalties.*

Establishing project starting dates is the first step in exercising control over the schedules produced by RAMPS. This is done merely by specifying the *earliest time period* at which work may begin on each of the projects to be scheduled.

The desired completion date is expressed in the same manner as the starting dates—the *time periods* during which the projects are to be completed. In many instances, it is difficult to provide a knowledgeable estimate of how much time, from start to finish, a project should require, particularly when the work is unprecedented. In others, experience will allow reasonably accurate completion dates to be set.

By providing RAMPS with the costs incurred in delaying each project, we can identify those projects with the higher priorities and thus provide the basis for deciding which projects are to be completed on the deadlines indicated if all targets cannot be met.

Control Factors

While the start date, completion date, and delay penalties provide a means for expressing priority among projects, there is also a need for expressing other criteria that are to be observed in the schedules that are produced for all projects. For example, in addition to recognizing the relative importance of projects, management may also recognize a need to minimize project costs, minimize idle resources, or maximize the total number of activities being worked on at all times. These and other needs can be conveyed to RAMPS through the use of the management control factors.

It should be emphasized that RAMPS attempts to complete each project as soon as possible by making the most efficient and appropriate use of time and resources. More specifically, RAMPS continually strives to:

1. Start and complete each activity at the earliest possible time.
2. Achieve a smooth rate of work accomplishment and resource utilization by "looking ahead" for possible bottlenecks.
3. Minimize idle resources.
4. Work simultaneously on as many activities as possible.
5. Give priority to critical activities.
6. Avoid interrupting work on an activity once it has been started.

In forming a schedule of work during *a given time period,* RAMPS is almost always confronted by conflicts among these objectives; rarely can they all be achieved at the same time. When there is conflict, the program must determine which of these objectives are to be satisfied at the necessary expense of excluding others. For example, consider the following situation: there are four activities that could be started in the current time period and there are adequate resources at the *normal utilization rates* to start all of them. However, one of the activities is on the critical path and in order to meet the desired project completion time, it must be scheduled at the speed-up rate. This will mean delaying one of the other projects so that the resources it would otherwise consume can be diverted to the critical activity. In addi-

tion, another activity must be completed during the current time period to avert a probable work bottleneck in the subsequent time period. To avoid the bottleneck, a second activity must be delayed so that the activity preceding the possible bottleneck can be accelerated.

Of the four activities that could be started, only two can begin if a work bottleneck is to be averted and a critical activity is to be given priority. The decision to be made is whether the first and fourth objectives are *more important* than the second and fifth objectives.

Certainly these considerations and decisions are common to any scheduling system. Indeed they are made every day by one means or another. In most instances, however, they come from an individual who is thoroughly familiar with the project being planned and is therefore able to judge the *relative importance* of each of the considerations.

The control factors that are a part of RAMPS offer a means for expressing the relative importance of the many items that bear on scheduling decisions. Through their use, those who are most familiar with the work to be done and the environment in which it will be done, can provide RAMPS with a means for determining the best course of action when there is a conflict in the overall scheduling objectives.

Thus, the discretionary use of management controls in the situations we have just discussed exercises a direct influence and control over the schedule produced.

Weighting the Control Factors

Influence of RAMPS' scheduling decisions is exercised by preparing a table of relative weights for the following six factors:

1. Free Float
2. Total Float
3. Look-Ahead
4. Work Continuity
5. Number of Jobs Scheduled
6. Idle Resources

As shown in Figure 6, each of these factors can be used to exercise a unique influence on the scheduling functions and thus satisfy one of the management objectives mentioned earlier in this section.

The degree to which the schedule is affected by any one factor depends upon its weight relative to the weights assigned to the other factors.

CONTROL FACTOR	EFFECT ON SCHEDULE	MANAGEMENT OBJECTIVE SATISFIED
Total Float Free Float	Gives priority to those activities which cannot be delayed without delaying project completion.	• Minimizes Project Completion Time • Gives Priority to Critical Activities
Look-Ahead	Gives priority to those activities upon whose completion many activities are waiting.	• Avoids Work Bottlenecks
Work Continuity	Gives priority to those activities which were started earlier and are incomplete.	• Avoids Activity Work Interruption
Number of Jobs	Gives priority to starting as many new activities as possible.	• Maximizes the Number of Job Being Worked On
Idle Resources	Gives priority to assigning all available resources.	• Minimizes Idle Resources

Figure 6. Control Factors.

For example, if the paramount consideration in accomplishing the work on a project is keeping idle resources at a minimum, the idle resource factor might be assigned an over-whelmingly high weight in relation to the weights assigned to the other factors. This would indicate to the system that highest emphasis is to be placed on minimizing idle resources in making scheduling decisions.

Using the Control Factors

RAMPS first examines the three utilization rates prescribed for each of the competing activities and lists all the possible assignment combinations. The weighted control factors are then consulted in deciding which combination is the most desirable. In evaluating the possible assignments that can be made, RAMPS must also consider the indicated project completion dates, project delay penalties, and activity interrupt penalties.

Occasionally, these items override one or more control factors. This might occur, for example, when a project has a high delay penalty so that it becomes imperative from a cost standpoint to give priority to critical activities even though the free and total float control factors carry a relatively small weight. In another instance, an activity with an extremely low interrupt penalty may be interrupted even though the interrupt control factor is shown to be most important. However, the management control factors are used in scheduling *every* time period and will therefore dominate sched-

uling decisions and directly control the schedule produced.

Four of the control factors, free float, total float, work continuity, and look-ahead, are used to evaluate individual activities that are competing for a resource during the time period being scheduled. These controls are used in deciding which activities are to be scheduled, and how many units of resource they are to receive.

The remaining two factors, number of jobs and idle resources, pertain to evaluation of all the assignments contemplated for the period. They ask the questions: "Does this assignment pattern minimize idle resources? Does it assure work on as many activities as possible?"

Total Float

The total float factor is intended primarily to place special emphasis on scheduling those activities with little or no total float. Therefore, the *smaller* the total float for an activity, the greater the emphasis placed on fulfilling its resource demands. Because activities on the critical path have no total float, a high total float weight forces early project completion because those activities on the critical path are given highest priority. After the critical activities have been served, activity priority decreases as total float increases.

Free Float

The free float control allows priority to be given to those activities with little or no free float. It therefore serves a two-fold purpose:

1. Expedites project completion by giving priority to activities on the critical path, and
2. Expedites activities with interfering float and thereby avoids bottlenecks that can occur when there is more than one critical path.

Because it gives priority to those activities that are likely to become critical in later time periods, the free float control can be an effective defense against delay in circumstances that demand that projects be completed as early as possible.

Look-Ahead

The look-ahead control provides a short-range guard against undesirable work stoppage or slow-down by giving priority to those activities upon whose completion several other activities are waiting. By means of the look-ahead control, RAMPS takes early preventive action against work build-ups later that could possibly delay completion of the project.

We can also see that the look-ahead control can be extremely powerful in the production of a schedule that calls for completion of the projects in a minimum amount of time. The overall effect is a maximum utilization of available resources in the production of a schedule that calls for a minimum project completion time.

Work Continuity

In every multi-project scheduling operation there are certain to be several activities that cannot be interrupted without incurring extra costs. Some activities might have high start-up costs; others might involve work with perishable goods which would be ruined if work were interrupted. In instances such as these, the work continuity factor is used along with activity interrupt penalties to show the importance of sustaining work on certain activities once they have been started.

In effect, a weight assigned to the continuity factor tells RAMPS that there are certain activities to be scheduled that should not be interrupted. The interrupt penalties assigned to these activities provide an indication of the relative costs of interrupting these activities. It is important to understand that activities with *no interrupt penalties* are most vulnerable to interruption; all activities are vulnerable to interruption, whether or not they carry an interrupt penalty, if the interrupt factor has no assigned weight.

When the possibility of interrupting an activity exists, RAMPS weighs the value of the interrupt factor *and* the individual interrupt penalties for the activities involved to determine the feasibility of interruption.

Number of Jobs

The purpose of the number-of-jobs control is to maximize the total number of activities scheduled during each time period. To achieve this purpose, RAMPS tends to schedule many activities at the slow-down rate to allow limited resources to be spread thinly over many activities.

The overall effect of this control is as follows: 1) many activities are started at their earliest

possible start dates, 2) speed in completing activities becomes secondary to concurrent work on all activities during a given time period, 3) the schedules for the projects are extended over a wider span of time because of the slower work rates.

Because the number-of-jobs control sacrifices project completion speeds for ability to work on the highest possible number of jobs per time period, it is most applicable in situations that require sustained, widespread work at the expense of fast completion of activities and projects.

Idle Resources

The idle resource control is concerned with utilizing a high percentage of the available resources during each time period. It is, of course, useful for emphasizing the high importance of keeping the number of idle resources at a minimum during each time period. In most cases, men, machines, and other resources represent a continuous cost, whether or not they are actually working. The aim of the idle resource factor is to "work" all resources at all times.

ANALYZING THE RESULTS

General Description

RAMPS produces two major reports, a work schedule for each project and a summary of units of resource used each time period by resource type. Analyzing these reports to determine whether or not the desired results have been obtained involves answering such questions as, "Have the desired completion dates been met? Have idle resources been kept to a minimum? When and where are the various resources used and how many premium units have been allocated?"

The Work Schedules

Figure 7 shows the work schedule for Project A and a breakdown of the various types of information contained in all RAMPS schedules. Although RAMPS internally interrelates the schedules for all projects, the printed schedules are produced separately for each project. This allows those who are interested in a particular project to receive only the information for that project.

The heart of the schedule is the right-hand portion which shows the time periods during which each activity is to be worked and the quantity of each resource allocated during each period. To the left are the event numbers, activity names, resources required, the three utilization rates, and amounts-of-work—all of which have been reproduced from the original data given to RAMPS.

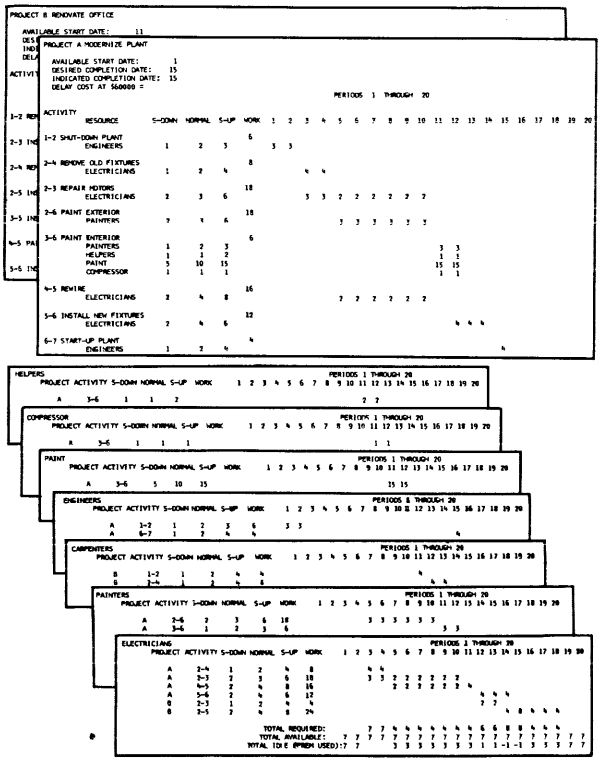


Figure 7. Output Schedules and Resource Allocation Summaries for Projects A and B.

The information in the upper left-hand corner of the schedule includes the specified start date, desired completion date, indicated or scheduled completion date, and the delay cost, if any. Obviously, this information is a key to determining the adequacy of the work schedule.

The Summary of Resources Allocated

Supporting the project schedules are reports showing how much of each resource was used during each time period and the activities to which the resource was assigned. These reports also include the number of units available, assigned, and idle in each period. Premium resources are indicated as negative idle units.

Note that each of these reports covers all allocations of a resource to all the projects.

When favorable answers appear, the reports form a springboard for setting the work plans in motion because they can be quickly turned over to supervisors, buyers, managers, and others throughout the organization who will be directly or indirectly responsible for getting the work done, obtaining the needed resources, and monitoring the progress of the work.

Perhaps of more importance, however, is how the reports are used to seek out flaws in the work plans that have caused the initial results to be disappointing or unsatisfactory. "Why was the target completion date missed? Should available resource quantities be increased? Decreased?" In large projects, there may be oversights in planning that create problems in the schedules. In many instances, they can be quickly corrected and the schedules put to immediate use; others will require extensive changes to the plan that make new schedules necessary.

It must be remembered, too, that because the information used by RAMPS is derived from the individual thinking of many people and based on estimates, it is fallible. However, one of the great advantages of RAMPS is that it can illuminate mistakes in planning before they are turned into wasted time, misspent funds, and misused manpower and materials.

ACKNOWLEDGEMENT

The authors wish to acknowledge the major contributions to the development of the concept, the basic algorithm and the operating program of RAMPS to W. Riley III and W. Dorfman as members of the C-E-I-R professional staff. G. J. Fisher, E. I. du Pont de Nemours and Company, Inc., was instrumental in proposing the original problem leading to RAMPS and was responsible for many suggestions that were eventually incorporated into the system.

REFERENCES

1. BELLER, WILLIAM, "PERT's Horizon Beginning to Widen," *Missiles and Rockets 9,* July 17, 1961, pp. 110-16.
2. BOEHM, GEORGE A. W., "Helping the Executive to Make Up His Mind," *Fortune 65,* April, 1962, pp. 128-31 ff.
3. CHRISTENSEN, B. M., "How to Take Guesswork Out of Project Planning," *Iron Age 188,* August 3, 1961, pp. 67-9.
4. *DOD and NASA Guide—PERT Cost—Systems Design,* Washington, Office of Secretary of Defense and National Aeronautics and Space Agency, 1962.
5. FRISHBERG, M. C., "Least Cost Estimating and Scheduling—Scheduling Phase Only (LESS)," Los Angeles, International Business Machines Corp., n.d.
6. HUDSON, JAMES P., "Program Evaluation Procedure. A Description of the WWDC PEP Program," Wright-Patterson Air Force Base, Aeronautical Systems Division, n.d.
7. KELLEY, JR., J. E., "Critical Path Planning and Scheduling. Mathematical Basis," *Operations Res. 9,* 1961, pp. 296-320.
8. KELLEY, JR., J. E. and WALKER, M. R., "Critical Path Planning and Scheduling," *Proceedings of the 1959 EJCC,* National Joint Computer Committee, 1959, pp. 160-73.
9. MILLER, R. W., "How to Plan and Control with PERT," *Harvard Bus. Rev. 40,* March, 1962, pp. 93-104.
10. *PERT and Companion Cost System—Handbook,* Washington, NASA, 1962.
11. VORESS, HUGH E., *et al.,* "Critical Path Scheduling—A Preliminary Literature Search," U. S. Atomic Energy Commission, Report No. TID-3568(Rev. 1), 1962.
12. YOUNG, L. H., "Now Industry Schedules by Computer; PERT (Project Evaluation and Review Technique) or Critical Path Method (CPM)," *Control Engr. 9,* January, 1962, pp. 16-18.

TIME-SHARING ON THE FERRANTI-PACKARD FP6000 COMPUTER SYSTEM

M. J. Marcotty, F. M. Longstaff & Audrey P. M. Williams
Ferranti-Packard Electric Limited
Industry Street
Toronto 15
Ontario

INTRODUCTION

A major advance in the design of computers was the provision of buffered or autonomous peripheral transfers which enabled computation to proceed concurrently with a transfer. In many computations, however, the total saving obtained is quite small; for instance, 10 milliseconds of computing overlapped with 100 milliseconds of transfer provides a gain of only 10% and the central processor must remain idle for the remaining 90 milliseconds. An extra burden is also placed on the programmer who must arrange his work so that computation can take place while the transfer is occurring. This is enough of a task when the program is written in machine language but, when the program has to be compiled, the compiler has to be considerably more sophisticated in order to reach the same efficiency and, as a result, compilation will take longer. The problem boils down to an economic one of weighing the cost of program production against the cost of machine time gained by the extra efficiency.

One step away from this dilemma is to use magnetic tape as the sole input and output medium for the computer and to provide off-line equipment for the transcription of data to or from magnetic tape. The transfer rate of magnetic tape is much higher than that of punched cards or printers and the loss of time during transfers is thus reduced. This method, however, requires extra off-line equipment which may be expensive and lack flexibility.

An alternative approach is to enable the central processor to store several programs simultaneously, and, each time a program is forced to wait for a peripheral device, to enter an alternative program which is at that time able to proceed. In this way the efficiency of the overall installation is increased since the central processor is kept working for a higher proportion of the time and a larger set of available peripherals can be kept operating. This method does not require any special off-line equipment and is very flexible in use.

In order to obtain system efficiency, one method to adopt might be to arrange, at the time of programming, for a number of jobs to run concurrently. The problems encountered with this method are connected with implementation and are the same as those found with a single program system using autonomous transfers, but increased in complexity by several orders of magnitude. Once the marriage between the several jobs has been consummated, it is practically indissoluble and the same group of jobs always has to be run at the same time. This may be perfectly satisfactory in certain special applications but generally this method is found to be too restrictive.

Alternatively, programs can be provided with suitable latching arrangements, so that,

although written separately, they may be joined together to make an efficient entity. However, the programmer still has to be very conscious that his program will be associated with a number of others when being run and has to imbue his program with a social instinct, which is tedious and not always easy to do.

Therefore, in spite of some loss of central computer time in order to perform organizational functions, it seems advisable to be able to write programs with no thought that they may be sharing the computer with other programs and to have a supervisory routine to make all the switches between the programs. This system of operation we refer to as "time-sharing."

As an example of time-sharing on a computer, consider the example of three programs which are, in order of priority:

Program P1. A routine which reads data from punched cards and stores it on magnetic tape, the contents of two cards being stored in a single tape block.

Program P2. A routine performing a magnetic tape sort with a simple keyword.

Program P3. A routine performing a long calculation using no peripheral equipment at this stage.

The basic rule of operation is that the program with the highest priority is entered whenever possible. A program which is waiting for a peripheral transfer to be completed is said to be "suspended." In Diagram I a time chart is shown which demonstrates how these three programs share the time of the central processor one with another. In order to demonstrate as many program interactions as possible, the time scale has been deliberately distorted.

In a recent paper,[1] it has been stated that a minimum of five requirements must be provided in order to have an operational time-sharing system, unless it is to be restricted to jobs which are specifically designed to be run concurrently or to fully debugged programs (but, except for trivial cases, it is impossible to know when programs are fully debugged since they are similar to scientific theories—they can only be disproved). These requirements are:—

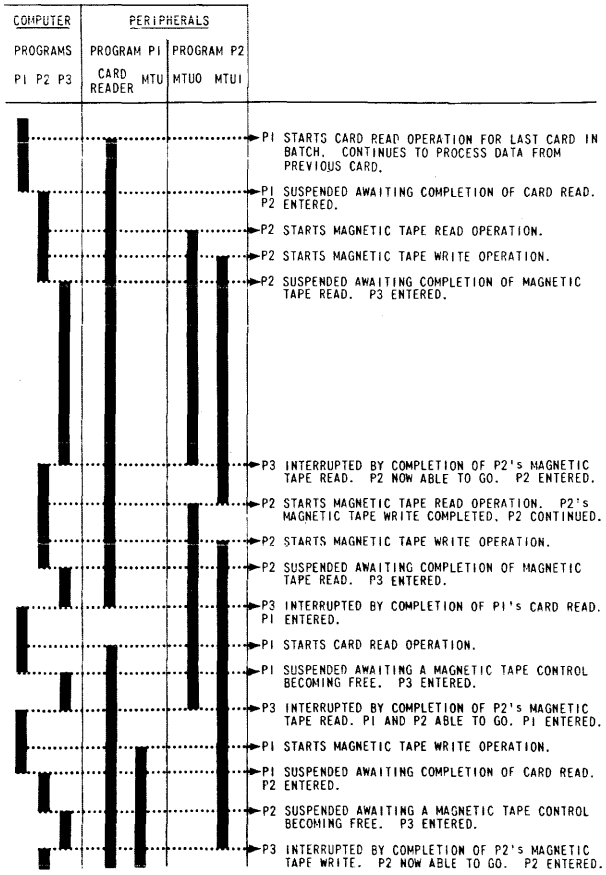


Diagram I.

1. Memory protection to prevent one program from destroying others.
2. Program and data relocatability so that the same routine can be used in different locations at different times.
3. A supervisory program.
4. An interrupt system.
5. Symbolic addressing of peripheral devices.

In this paper we hope to show how these requirements have been met in the FP6000 system by means of a combination of hardware and software.

THE FP6000 COMPUTER SYSTEM

The FP6000 is a fast medium sized computer system constructed in a packaged, modular form. At the centre of the system is a general purpose digital computer with a minimum core store of 4096 words which may be increased in modules of 4096 words to a maximum of 32,768

words. The machine operates in the binary mode and a word, of length 24 bits, can be used to represent an instruction, a signed integer or fraction, or four 6-bit alphanumeric characters. The store cycle time can be either 2 or 6 microseconds. The computer is a parallel machine with a clock rate of one megacycle.

A large variety of peripheral devices can be added to the computer, making the system versatile and adaptable to commercial data processing, scientific and real time applications. The peripheral equipment is broadly divided into two classes: character at a time, for example paper tape readers, and word at a time, for instance magnetic tape. For each type of equipment in either one of these two classes, the interface between the peripheral device and the computer is made the same. Due to this modular method of attachment, it is easy to expand any system to meet increasing demands and wider applications.

A program called EXECUTIVE organizes all peripheral transfers and the allocation of the central processor's time to the various programs being run simultaneously. This program is written in such a way that the programmer does not have to consider time-sharing or the mechanism of peripheral transfers when writing programs. In addition, EXECUTIVE provides certain macro-instructions for carrying out floating-point arithmetic and enabling the programmer to have a master program which time-shares with one or more sub-programs.

The system is controlled from a console which has deliberately been made simple. Basically, it consists of an electric typewriter which enables the operator to give instructions to EXECUTIVE and EXECUTIVE to pass messages to the operator. A large number of these messages are concerned with manual action which is required on the peripheral equipment. In addition to the typewriter, the console has six buttons: to call the attention of EXECUTIVE to an incoming message on the typewriter, to cancel such a message and so on. A paper tape reader and punch, when incorporated in a system, are also located on the console.

THE CENTRAL PROCESSOR

The core store of the central processor is not divided into blocks in any way; however, each program in the machine at any one time has a definite area of store allocated to it. A region of core store is allocated to a program by EXECUTIVE; this is done by specifying two addresses, the starting address known as the "datum" and the final address called the "limit." The core store area between a program's datum and limit is called the program's reservation. Datum and limit points occur only at multiples of 64 words. These addresses are set by EXECUTIVE and stored in special registers in the arithmetic unit during the time a program is being obeyed. In operations where addresses of locations are used, datum is added automatically by hardware to the addresses; because of this, programs are written with addresses relative to the datum and, therefore, are locatable anywhere in the core store. Indeed, during the running of a program, EXECUTIVE may transfer the program to another area of core store in order to make room for a new program. All that needs to be done after the transfer from one area of core store to another is for EXECUTIVE to change the values of datum and limit corresponding to the program. This operation of relocation is only carried out after EXECUTIVE has ensured that no peripheral transfers are in progress.

The first eight core store locations belonging to each program are known as "accumulators" and can be used for arithmetic and counting. Accumulators 1, 2 and 3 may also be used for indexing.

Programs are stored with one instruction per word; for a normal instruction the word is divided as shown:

X	F	M	N

where X is an accumulator address
F is a function code
M is an index register address
N is a core store address or a number.

Obeying normal instructions consists of performing the function F between the contents of the core store location N and the contents of the accumulator X and placing the result in either N or X depending upon F.

There are 16 groups of 8 function codes, though not all function codes are allocated. Five of these groups consist of macro-instruction codes: these operations are performed by EXECUTIVE. While a function is being obeyed datum is automatically added to the appropriate addresses in the arithmetic unit and checks are made that these addresses do not lie outside the program's reservation. These checks are performed by special hardware in the arithmetic unit referred to as the "reservation checker." Since a program cannot be allotted less than 64 words there is no necessity to check that the accumulators are within the program's core store area. However, the *N*-address is so checked.

The operation of a simple order takes place in five phases or beats:

Beat 1.
During this beat the address of the current instruction is sent to the core store and the instruction is obtained and placed in the arithmetic unit. The contents of this core store location are regenerated so that no change is made to the store by this operation. However, if at the end of the beat, it is found that the address of the current instruction did not lie in the area allotted to the program, further accesses to the core store are inhibited, though the remainder of the instruction proceeds as usual.

Beat 2.
If the *N*-address of the instruction requires indexing, the value of the index register is taken from core store, regenerated, added to the address and the result stored in the arithmetic unit. This beat is omitted where indexing is not required.

Beat 3.
The first of the two operands required for the function is taken from the core store and put into the arithmetic unit. It is arranged that the first operand to be taken from the core store is the one which is unchanged by the function; that is, after being read out it is regenerated in the core store. At the end of this beat, the result of the reservation check on the indexed *N*-address is examined and, as at the end of beat 1, if the check has failed, further accesses to the core store are prevented.

Beat 4.
The second of the two operands is obtained and, since its value is to be changed, it is not regenerated. The function is performed during this beat.

Beat 5.
The result of the operation is written back into store and, except in the case of branch instructions, the address of the current instruction is increased by one. This address is retained in the arithmetic unit ready for the next instruction.

Then follows the beat 1 of the next instruction. If the reservation check has failed at either of the two points where it was tested a forced entry to EXECUTIVE is made at this time.

There are two modes of operation of the central processor: the Normal mode and the Executive mode. The Executive mode, as its name implies, is used while EXECUTIVE is being obeyed. While in Executive mode, under the control of EXECUTIVE, the current value of the datum may be added to any or none of the addresses *N*, *X* and *M* in an instruction. This enables EXECUTIVE to have access to information within a program's area, using the program's index registers when required. Reservation checking is not performed during the operation of EXECUTIVE and some of the functions in the group of macro-instructions have special actions, though most are unassigned.

The above example of an entry to EXECUTIVE, due to a reservation check failure, might be termed an "involuntary" entry or "interruption." The other reasons for an involuntary entry are:

(a) Depression of a console push button.

(b) An illegal order; that is, one with an unassigned function code.

(c) A monitor point. In order to assist a programmer in developing a program, EXECUTIVE can arrange for an interruption to occur after certain specified types of instruction, for example a successful branch instruction, in a particular program.

(d) The time register has reached zero. This register, which is an optional extra, enables time-accounting to be performed in a service-center machine or interruptions to occur at specific intervals so that an operation can be performed on a special peripheral device.

(e) A peripheral incident, which can be an indication that either a peripheral transfer has been completed or one of the automatic checks on the transfer has failed.

When an involuntary entry to EXECUTIVE occurs, the current instruction of the program being interrupted is dumped in location 8 of that program. The state of the overflow register and certain other indicators particular to the program being left are also stored in the same location at this time, so that, when a return is made to the program, the state of these may be reset correctly. A reason for entry register is set so that EXECUTIVE can find out the reason for the interruption. The machine is then set to Executive mode and the instruction number in the instruction number register is set to the entry point in EXECUTIVE corresponding to an interruption.

When a program is input to the computer a priority number is assigned to it. This number is allocated by the programmer and is specified on the program tape with a number of other parameters such as the amount of core store space, drum space and other peripheral equipment required by the program. The priority number is originally assigned on the basis of the number of peripherals used by the program and the proportion of time spent in computation relative to that awaiting peripheral transfers.

A maximum of four programs may be run simultaneously; however, this is an arbitrary limit set by EXECUTIVE and is not dictated by hardware considerations. When EXECUTIVE was being written, in order to assign store space for certain lists, such a limit had to be established, and it was felt that four was a reasonable number for a machine of the size of FP6000. Since this limit is set by EXECUTIVE, there is nothing to prevent, where the installation warrants it, a special version of EXECUTIVE being assembled with a higher limit. Each program may be divided into a master and at most two sub-programs, which are each assigned an individual priority. EXECUTIVE keeps a record of each program and sub-program, in descending priority sequence. This list is called the Priority List and has at most twelve entries. Also recorded in the Priority List is the present state of each program: free to be obeyed (called Active), suspended awaiting a peripheral, suspended awaiting an operator message, and so on. Whenever EXECUTIVE has completed its required actions, it scans the Priority List for the active program with the highest priority and transfers control to this program. Each program is given a code name which is made up of four alphanumeric characters and is used in all communications between the operator and EXECUTIVE by means of the console typewriter.

During input of a program, EXECUTIVE allots the core store to be assigned to the program according to the core store requirements given on the program tape and the core store available in the computer at the time. In order to keep as large as possible an area of contiguous core store locations free, whenever a program is finished and is no longer required in the computer (the term used is "abolished"), all current programs stored above the program to be abolished are moved down the core store to fill up the now vacant space. When the core store area has been assigned to a program, the appropriate values of datum and limit are stored among the records which EXECUTIVE keeps for each program currently in the computer.

When referring to peripheral units in a program, the programmer numbers his units of a particular type starting from zero. Suppose, for example, that the program named "BILL" uses four magnetic tape units. The programmer will number these units 0, 1, 2 and 3. If BILL is to be run on an FP6000 system with six tape units which are numbered 9 to 14 and if, at the time when BILL is input, units 9 and 12 of the system are already being used by current programs in the system, then EXECUTIVE will allot units 10, 11, 13 and 14 to BILL. EXECUTIVE will type out a message to the operator giving the correspondence between BILL's unit numbers and the system unit numbers. Every time BILL refers to his magnetic tape unit 0, system unit 10 will be used. Because of this arrangement for the numbering of peripheral units, all peripheral transfers are handled by EXECUTIVE. This method also shields the programmer from a lot of the work in organizing peripheral transfers. If BILL is run on another occasion, the allocation of actual units to the units 0 to 3 in BILL may be quite different, but, since the change from the program's

unit numbers to the system unit number is performed automatically by EXECUTIVE, no alterations have to be made to BILL.

ATTACHMENT OF PERIPHERAL EQUIPMENT

In general, each peripheral device has its own control unit; exceptions to this rule are magnetic tape and drum units, several of which may be connected to one control unit. Each control unit has the address of a special-register assigned to it; this special-register performs the function of a mail box for the passing of control information to and from the control unit. In addition, each control unit has a core store word associated with it which is used by the peripheral control unit to store the transfer address and the count of words or characters to be transferred. The initial value of this control word must be set prior to a peripheral transfer being started. The operation may then be initiated by sending a starting signal to the special-register for the device. The function for sending instructions to the special-registers is available only to EXECUTIVE.

Once a transfer has been started it proceeds autonomously, each word or character being transferred to or from the core store as requested by the peripheral control unit. Between such transfers the central computer is free to obey programs. When a request for the transfer of a word or a character occurs it competes with similar requests from other peripheral units and, eventually, the transfer takes place. There is nothing new in this approach; several of the larger computers use this method. However, the usual system is to have a separate piece of logic in each peripheral control unit to perform the addressing and counting operations required, the central processor thus being able to continue with its normal operations, unhindered except when the peripheral control unit requires access to the core store at the same time as the central computer. When this occurs, a small hesitation in the rhythm of the central computer takes place. Since FP6000 is a medium-sized computer it was decided that, in order to reduce the cost, it would be better to use the central arithmetic unit to perform the addressing and counting functions. The actual amount of time spent by the arithmetic unit in performing hesitations remains a small proportion of the total time required for the transfer; for example, with a 2 microsecond core store, during a magnetic tape transfer, the arithmetic unit will be available for program calculation for at least 90% of the time.

When a peripheral control wishes to make an access to the core store it sends a hesitation request signal to the computer control unit. If more than one of these signals occurs at once, a priority system ensures that the faster peripheral units get precedence over the slower ones. The chosen peripheral control is sent a hesitation select signal and, at the end of the instruction currently being obeyed, the next instruction is not started but a hesitation is performed. A hesitation requires 4 beats:

Beat 1.
During this beat the core store address of the peripheral control word is supplied to the core store and the control word is read into the arithmetic unit. A counting operation is performed and, if this is the last hesitation in the transfer, a stop signal is sent to the peripheral control unit. The address part of the control word is augmented to give the core store location to which data is to be transferred during the next hesitation.

Beat 2.
The new value of the control word is written back into the core store.

Beat 3.
The address from the control word is supplied to the core store and the word is read.

Beat 4.
Depending on the direction of the transfer, either the new data word is written into the store or the original data word is regenerated.

Then follows the next order or another hesitation. Magnetic tape units attached to FP6000 have a character transfer rate of 65 KC/S—that is, one character every 15 microseconds or, since magnetic tape transfers occur one word at a time, one word every 60 microseconds. The net interval is a fraction of this if several magnetic tape units are performing transfers simultaneously. Some of the instructions in FP6000 take longer than 60 microseconds to be obeyed; provision has therefore been made for hesitations to be carried out, if required, at certain defined break-points within the sequence

of basic operations which make up an instruction, as well as between instructions.

The only instructions which have to be broken into to make a hesitation are the longer ones which nearly always involve some sort of loop in the micro-program (for instance multiplication or division), and therefore a breakpoint is arranged in the loop. During the hesitation process only two of the registers in the arithmetic unit are used. If a hesitation has to be made, the contents of these two registers are stored temporarily in core store locations 9 and 10, which are set aside for this purpose. Diagram II presents a simplified flow diagram of the operation. From this it will be seen that only hesitation requests from peripheral controls which cannot wait, fast hesitations, are allowed to interrupt the course of an instruction. When the hesitation operation is completed, and if there are no more fast hesitation requests, the two registers are reset from core store and the long instruction is carried on from the point at which it was suspended.

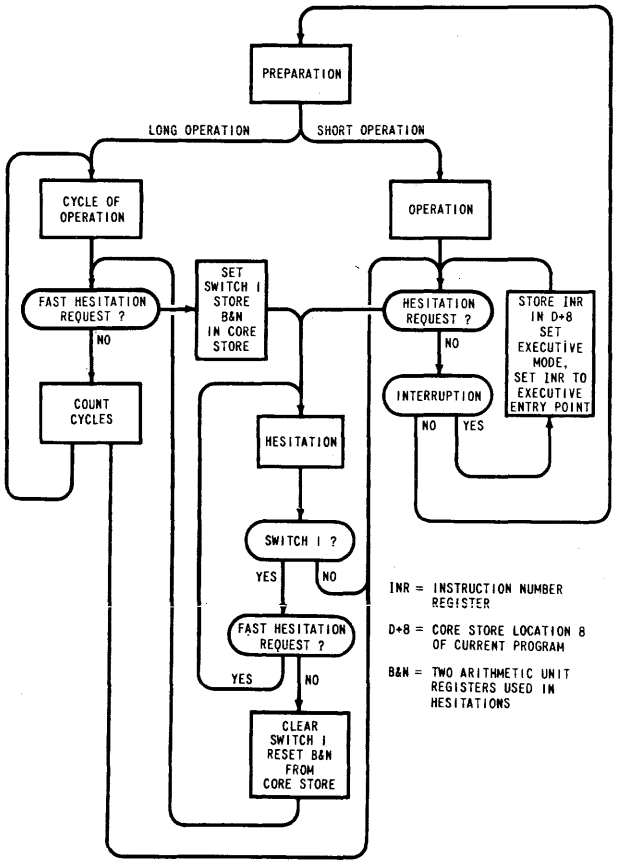


Diagram II. Simplified hesitation flow diagram,

During the course of a hesitation no reference is made to the output of the reservation checker; to do so would involve making suitable settings in arithmetic unit registers, which would be too costly in time. EXECUTIVE therefore checks, before initiating a transfer, that the transfer does not involve any core store locations outside the area of the program calling for it. This test is very simple to make and involves taking the starting address for the transfer, adding to it datum and the number of words to be transferred, and checking that the result is less than the value of the limit.

If, during a hesitation, a stop signal is sent to the peripheral control unit because the end of the transfer has been reached, the peripheral control unit will terminate the transfer and, when this has been done, send a signal to the central computer. This signal causes an entry to be made to EXECUTIVE at the end of the instruction currently being obeyed. This type of entry may also be classed as an involuntary entry. When such an entry is detected by the central control unit a mark is made in the reason for entry register. This consists of a special register which may be read by an instruction available to EXECUTIVE only. Corresponding to each of the reasons for entry—depression of a console push button, illegal order or reservation check failure, monitor point or the time register reaching zero—there is one bit in this register. The peripheral incident type of entry is expanded so that there is a bit which corresponds to each peripheral control unit. Before entry is made to EXECUTIVE the current value of the instruction number register is stored in core store location 8 of the program being obeyed at the moment of the interruption. The machine is then switched to Executive mode and the address of the entry to EXECUTIVE is forced into the instruction number register.

EXECUTIVE

Diagram III gives a flow diagram of the operations which occur during an involuntary entry. We will call this "EXECUTIVE Entry I."

The first action on entry is to read the reason for entry register; if this is not zero the cause for interrupt with the highest priority is isolated. The bits giving the reasons for interrupt

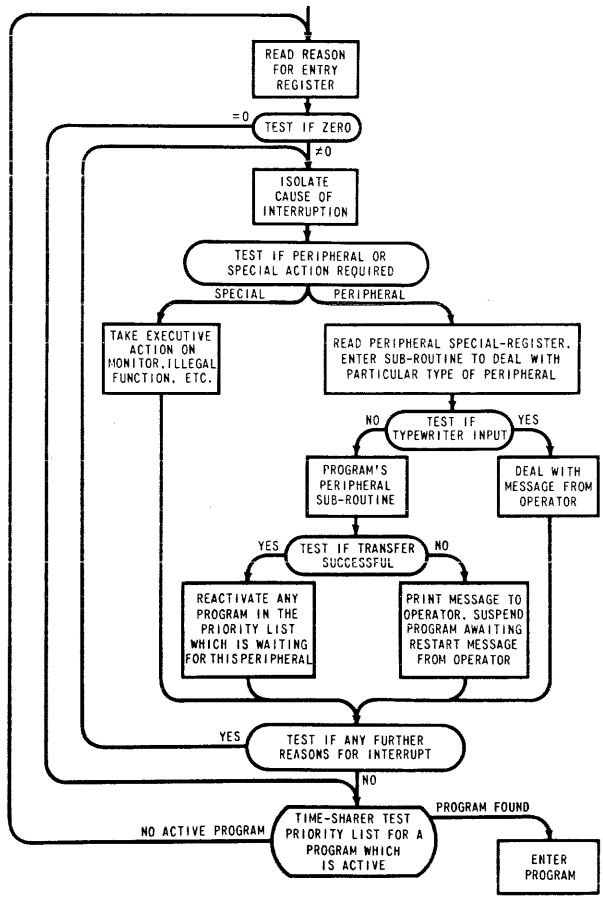


Diagram III. Executive Entry 1.

fall into two main classes: those involved with special action, for instance an illegal function, and those involved with peripheral transfers.

If the interrupt is one where special action is required the necessary action is performed. In the case of monitor action the program development routine is entered to print out the required monitoring information for the programmer. In the case of an illegal operation or a reservation violation the offending program is marked in the priority list as being suspended. When a program is suspended for either of these reasons it is marked in EXECUTIVE's records as awaiting a message from the operator. The operator is informed, by means of the console typewriter, that the program has been suspended and the reason for this suspension is also given.

If a peripheral incident is the reason for interrupt, the actual control unit may be identified from the reason for entry register. With this information, the special-register associated with the peripheral control unit may be interrogated. Since the reasons for an interrupt can vary with the type of peripheral, a separate subroutine is used for each type. For example, a paper tape reader can cause an interrupt, not only at the end of a transfer, but also if the parity of the character read from the paper tape is incorrect or if there is no paper tape to be read. In addition to the three bits used for signalling these three conditions there is also a bit used to signal the fact that the reader is busy. The subroutine which deals with the type of peripheral causing the interrupt examines these signals by reading the special-register associated with the peripheral control unit, thereby clearing the indicator bit in the reason for entry register, and tests if the transfer has been successful. If so, any program which has been suspended because it is waiting for this peripheral is reactivated. If the transfer has not been successful, the program making the transfer is suspended and a typewriter message is output.

In the case of the interrupt being due to the end of a message from the console typewriter, EXECUTIVE has to take action immediately on this message. The types of message which can be input on the typewriter are:

1. GO #BILL

This message will reactivate program BILL and allow it to continue from the instruction number contained in its core store location 8, where the instruction number at the time of interrupt was stored. There are two main uses for this instruction: to start a program which has just been read in, since it is normal to hold a program in suspension immediately after input to allow its data to be set up on various peripheral devices; and to continue a program which has been suspended awaiting some operator action, such as changing a tape reel.

2. GO #BILL AT 1234

This message will reactivate program BILL and allow it to continue from instruction number 1234. This allows a program to have optional entry points and may also be used to facilitate restart procedures following a peripheral failure.

3. LOAD #BILL ON *X*

This informs EXECUTIVE that a new program is awaiting input on peripheral unit *X*. Provided that there are not four current programs already in the machine and that at least 64 words of core store are free, EXECUTIVE will read the first part of the new program to determine whether the requirements for core store and peripherals can be met. In either case a message will be output and, if the new program can be accommodated, EXECUTIVE will read it in under a binary read routine. This input operation will be time-shared with the other programs. The peripheral units of the installation each have an absolute number by which EXECUTIVE identifies them; these numbers are prominently displayed on the units.

4. LOAD #BILL ON *X, Y*

The LOAD message may optionally request *Y* words of core store, in which case this request overrides the store request on the program tape. This version of the LOAD message is most useful for programs such as matrix schemes for which the store required varies considerably with the specific data to be used.

5. SUSPEND #BILL

This suspends program BILL awaiting a further typewriter message. This message is mainly used in an emergency, for example, if the program seems to have got out of control and is continuously reading the same section of magnetic tape.

6. DELETE #BILL

This abolishes program BILL from the store and lists. Normally a program will be terminated by a special instruction in the program and the typewritten message is only used in special cases such as the program having gone out of control or if the computer is required for an urgent job for which there is insufficient space.

7. ALTER #BILL AT 1234/*FXMN*

This changes instruction number 1234 of program BILL to *FXMN*. This provides a means of altering a single instruction during program development.

8. CONTINUE #BILL ON *X*

This enables a further part of a program to be read in or the input of a program to be continued after a peripheral failure.

9. MONITOR #BILL ON *X*

This message informs EXECUTIVE that a program development tape is to be read from unit *X*.

10. PRIORITY PRINT

This is a request to EXECUTIVE to print out the current contents of the priority list.

11. REVISE PRIORITY #BILL *YY*

This provides the facility of changing the priority rating of a program so that an urgent job can be accelerated, at the cost of a reduction in efficiency for the overall system.

When the input typewriter message or the peripheral incident has been dealt with, a test is made to determine whether there are any further reasons for interrupt. This is done by checking the copy of the reason for entry special-register retained by EXECUTIVE. If there are any further reasons the process is repeated. If there is none the Time-Sharer is entered. This scans the list of programs and enters the active program with the highest priority. If there is no program which is active, a loop back is made to read the reason for entry special-register. This loop will continue until a peripheral incident occurs to make a program active.

There is a second entry to EXECUTIVE: this is from a program entry by means of the macro-instructions. There are four types of these:

1. Peripheral transfers.
2. Organizational instructions.
3. Master and sub-program instructions.
4. Floating-point operations.

Diagram IV is a flow diagram of EXECUTIVE for this entry. When an instruction which falls into one of these categories is encountered in a program the control unit of the computer first stores the current instruction number in the program's core store location 8, then transfers the indexed value of the *N*-address to core store location 1 in EXECUTIVE

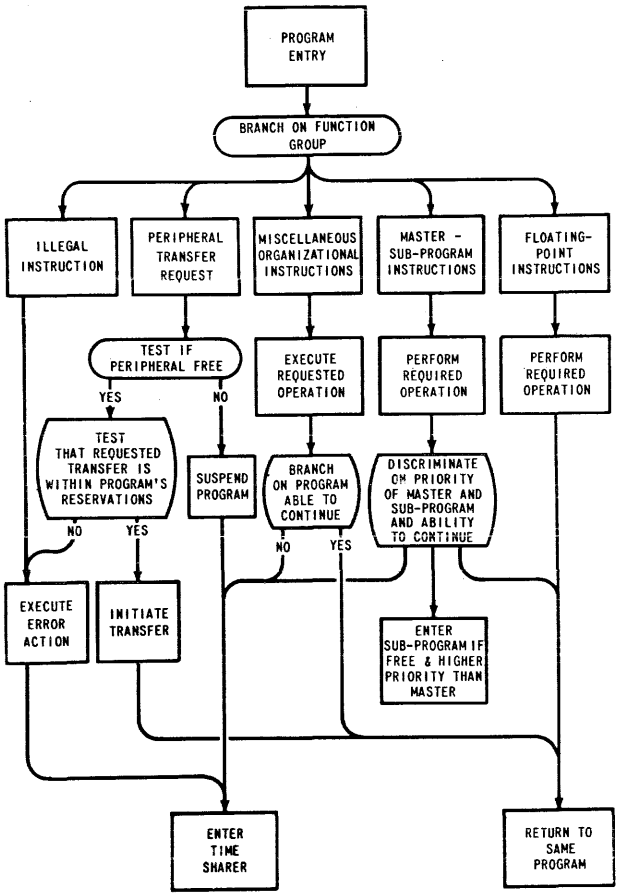


Diagram IV. Executive Entry 2.

core store area and transfers the instruction itself, not indexed in any way, to core store location 2. The machine is then set to Executive mode and the entry address to EXECUTIVE for a voluntary entry is forced into the instruction number register.

EXECUTIVE first examines the function which caused entry. Since not all the functions which cause entry to EXECUTIVE have been allocated, provision is made for an illegal instruction. In this case the program is suspended and a message to this effect is printed out on the typewriter.

For peripheral transfers, more information is required by EXECUTIVE than can be accommodated in one word. The actual transfer instruction is only one word, but the N-address (which may be indexed) specifies the address in core store of the control word, formed by the program, for the transfer. This word contains a counter of the number of words to be transferred and the core store address for the start of the transfer. The type of peripheral involved is defined by the function digits of the transfer instruction, and the particular program's unit number is given in the X-address. Before the peripheral transfer instruction is obeyed, though not necessarily immediately before, a mode of transfer must have been set by a special instruction provided for this purpose. This mode controls whether, for example, a reading or writing operation is to occur on magnetic tape. EXECUTIVE checks that the transfer lies within the program's reservation and that the program has, in fact, a unit X. If either of these checks fails the program is suspended and the operator informed. The unit is then checked to see if it is already busy, in which case the program is suspended waiting for the peripheral to finish its current transfer. If the transfer can be made, EXECUTIVE takes the information provided in the instruction, forms the appropriate control word and initiates the transfer. The program is then re-entered.

There are a number of instructions available concerned with the organization of peripheral transfers. As we saw just now, the initiation of a peripheral transfer does not cause automatic suspension of a program and it is up to the programmer to ensure that he does not cause any conflicts by neglecting this fact. The philosophy of this method is that it allows the programmer to arrange to be working with one set of data while the next set is being read or the previous set output and, at the same time, removes the need for costly hardware to check that a programmer is not using data before it has been read. An instruction of the form "Suspend this program if its unit X of type N is busy" is provided to enable the programmer to avoid conflicts.

Another instruction in the class of organizational instructions is "Suspend this program pending an operator message to EXECUTIVE." An instruction of this type will usually be preceded by the output of a message on the console typewriter, telling the operator of an action that is required on a peripheral unit (for instance, changing a reel of tape). The output of this message is under the control of the program. The program may, in this message, mention its own unit number or may ask EXECUTIVE what absolute unit number has been assigned on this occasion.

Apart from the facility of being able to perform time-sharing between independent programs, FP6000 is able, by virtue of EXECUTIVE, to have a program time-sharing with different parts of itself. We have already seen how, by use of the instruction to EXECUTIVE "Suspend this program if its unit X of type N is busy" a programmer is able to avoid overwriting information that is involved in a peripheral transfer. However, this may lead to the central computer being held up waiting for a transfer when in fact it could be performing useful work. Consider a program which is producing results which are to be printed on a line printer and suppose that, because of the nature of the computation, the results are produced in groups of ten lines, none of which can be printed until all are computed. This situation can easily arise in the production of certain types of table. If we adopt the straightforward approach, the sequence of events will occur as follows:

1. Compute group of results.
2. Print group of results.
3. Return to step 1.

Even if we assume that there are other programs in the machine which can use the arithmetic unit while the results are being printed, the printer is still idle for part of the cycle, so that it is not being used efficiently. This situation can be avoided by splitting the program into a master program and sub-program which time-share with each other. If we divide the core store originally allocated to the program into an area containing the master program, an area containing the sub-program and an area of working store, and also arrange that the datum and limit of the master program are such as to include the whole of the program's core store area and that those for the sub-program just include its own area and the working store area, the master program can be made to time-share with the subprogram. Because of the time-sharing ability of the computer, no computation can be timed absolutely, relative to a peripheral transfer. It is therefore necessary for the programmer to take suitable precautions so that it is not possible for overtaking to occur. The master program must not be allowed to overwrite results with new ones before the previous set has been printed, nor must the subprogram output a set of results before they have been completely processed. Similar situations arise with input sub-programs.

Three instructions are provided to enable the programmer to organize the time-sharing between his program and sub-programs without difficulty. When the master program reaches a point at which it is ready to initiate the sub-program, for example when it has some results which are ready to be printed, an instruction of the form "Activate my sub-program X entering at instruction N" is obeyed. The value of X can be 1 or 2, allowing a master program to have two sub-programs. This instruction causes EXECUTIVE to activate the sub-program, set the sub-program's instruction number equal to N and enter the time-sharing routine. Note that the sub-program is not necessarily entered at this time; it is simply activated within the priority list. Usually, because the sub-program is using peripheral equipment, the priority of the sub-program will be higher than that of the master. The master program remains active so that it time-shares with its own sub-program, and EXECUTIVE will pass control initially to the part with higher priority.

When the master program has a new set of results to be printed, but wishes to check that the output sub-program is ready for them, an instruction of the form "Suspend me if my sub-program X is active" is obeyed. If the sub-program is still actively engaged in printing the last set of results, the master program will now be suspended to enable the sub-program to catch up. If, however, the sub-program has completed its task at this stage, it will have made itself inactive by the instruction "Suspend me awaiting reactivation by the master program, and reactivate the master program if it is waiting for me." Thus, whether the master or sub-program completes its task first, eventually a stage is reached at which the master program is informed that the sub-program has finished and is inactive. The master program may reactivate the sub-program immediately, or it may wish to do some preliminary data transfers before initiating the next set of output.

A more sophisticated programmer may improve on the utilization of the printer by dividing his working store into three areas, thus buffering the supply of data from master to

sub-program. By means of markers set within the common area, the master and sub-program can communicate their relative states, and need only enter EXECUTIVE when it is necessary for one or other to be suspended. However, for most purposes the simpler approach will produce a satisfactory increase in peripheral utilization.

The remaining type of instruction which causes entry to EXECUTIVE is the floating-point group. These are macro-instructions causing floating-point arithmetic operations to be performed on operands by means of sub-routines in EXECUTIVE.

Since a system may have a variable number of peripherals and a variable number of programs, it is impossible to give absolute times for any EXECUTIVE operations. However, some sample times may give an indication of typical EXECUTIVE operations. With a 2 microseconds core store, a peripheral transfer, on a unit type of which there are 4 units in the system, can be initiated within 350-600 microseconds. The time taken to suspend one program and enter the next active program can vary between 300-550 microseconds. The size of EXECUTIVE varies with the configuration of peripheral types and with the variety of typewriter messages required by this particular system. The minimum size for EXECUTIVE, corresponding to a small system, is about 600 words. A system with an 8096 word core store and 4 peripheral types might have a 1200 word EXECUTIVE.

Since EXECUTIVE has been written as a set of routines which communicate with each other by means of codeword addresses, the EXECUTIVE required for a particular system may be assembled quickly. Similarly, if a new peripheral type is added to an installation, an instruction which previously was illegal is now routed to a new subroutine to perform the required transfer. The subroutine is simply added to the EXECUTIVE program at the end of its area. Since EXECUTIVE may have packages added or removed so easily, the individual FP6000 installation may decide for itself which facilities it wishes to have included at the cost of some core store. For example, it costs 40-50 words to permit ALTER messages from the typewriter. In a system where an established set of programs use most of the operating time, it may be felt that the ALTER facility is not as valuable as an extra 50 words of core store available to the programs.

In this paper we have tried to show the way in which the time-sharing facilities normally only available on larger computer systems may be realized on a medium-sized system by an intermarriage of hardware and software for minimum cost yet still retaining all the safeguards necessary for an operable system. We have shown how the five criteria given by Amdahl have been met and how, in addition, we have been able to provide other facilities designed to give a higher efficiency to the system.

REFERENCE

1. GENE M. AMDAHL: New Concepts in Computing System Design. Proc. IRE Vol. 50, No. 5, May 1962.

THE D825 AUTOMATIC OPERATING AND SCHEDULING PROGRAM

Rankin N. Thompson and John A. Wilkinson
Burroughs Corporation, Burroughs Laboratories, Paoli, Pennsylvania

INTRODUCTION

This paper concerns a general executive program for the Burroughs D825 Modular Data Processing System. The D825 is a large-scale, multicomputer, expansible, general-purpose, digital system employing automatic parallel processing and an extensive system of interrupts, and is especially suited to military command and control applications. Military designation is AN/GYK-3(V).

The implementation of the executive program, called the Automatic Operating and Scheduling Program (AOSP), was completed in the fall of 1962 and demonstrated at the 1962 Fall Joint Computer Conference (AFIPS). Since then, the AOSP has undergone testing with a variety of program mixes.

While the bulk of the AOSP concerns retrieval of named objects from files, run-time allocation of memory, and input/output (I/O) control, the primary AOSP function is to provide facilities for multicomputer multiprocessing—that is, the use of one, two, or more processors (computer modules) by time-sharing their arithmetic/control functions between one or more object programs under control of an executive system. In this way, not only may a program time-share a processor with other programs, but a processor may time-share the execution of a program with other processors. The programmer need not indicate that his program may be processed in parallel with other programs; the AOSP effects parallel processing of job entries as a matter of course. However, if a large program involves major independent subtasks, the specification of parallel operation *within the program* may be made by the programmer. Hence, a characterizing feature of the AOSP is that a program may invoke a parallel process. Another feature is that several processors may be—although none are aware of it—simultaneously executing the same set of instructions, referencing different data sets. This provision allows more efficient use of memory.

The assignment of processors and memory space to programs is accomplished through a program designed for execution in parallel on several processors. This program and its associated tables constitute the AOSP. The AOSP has embedded within it controls which prevent simultaneous assignment of a given equipment module to several tasks, or of a given task to several modules. A program has associated with it not a processor, but an entry in a job table which any processor may select for execution. When a processor suspends a program to service an interrupt, the values of its registers at the moment of interrupt are stored in the area from which they were originally loaded. Any processor, not necessarily the interrupted processor, may then resume the execution of the program.

Much of the AOSP necessarily functions "behind the back" of a user's program. The establishment of linkages between programs and data, the scheduling of deferred I/O re-

quests, equipment scheduling, and the response to system interrupt conditions are independent of any foreknowledge of the interrupted program. However, many of the AOSP functions are available to the programmer at his option, via *control macros,* which are essentially subroutine-like calls. Macros are used, for example, to locate a specified object in system files, to request parallel processing at an appropriate branch point, or to permit certain tasks to be suspended until all paths of a branched program reach a rejoining point.

The AOSP was designed to operate on D825 systems with various module configurations. Processors, memory, I/O control modules, and I/O devices may be added or removed by appropriate changes to system configuration tables.

BASIC D825 SYSTEM CONFIGURATION

The D825 Modular Data Processing System[1] consists of an arbitrary configuration of identical memory, I/O control, and computer (arithmetic/control processor) modules. The configuration may be organized as a combination of the following modules:

1 to 4 arithmetic/control processors.
1 to 16 memory modules (4096 words/module).
1 or 2 I/O exchanges.
1 to 10 I/O modules per exchange.
1 to 64 peripheral devices (drums, magnetic tapes, card readers, etc.) per exchange.

The memory complement in the D825 is fully shared—accessible by all processors and I/O control modules. Physically, the memory consists of a number of separate core memory modules. Each processor has exclusive use of a data transfer bus by which it can communicate with any memory module. There is no direct data transfer between processors and I/O control modules; processors are used for more suitable tasks during transfers between memory and I/O. The I/O control modules of an I/O exchange are connected to the memory modules by a single time-shared bus. The various peripheral devices are interconnected with all of the channels of the I/O exchange. All data flow between modules is effected as independent memory read or write operations by the processors or the I/O control modules. In addition, there are control lines between each module and all other modules which indicate status and provide clock synchronization.

A schematic diagram of a D825 system is shown in Fig. 1. The significant features of each module type of the D825 are listed in Table I. The design rationale which led to the organization of the D825 has been described by Anderson, *et al.*[1]

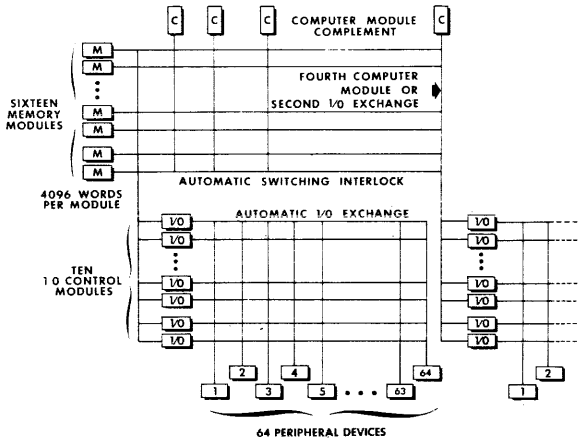


Figure 1. D825 System Organization.

THE OPERATING ENVIRONMENT

The AOSP is designed for use by a computing system operating in a command and control environment. As stated by Anderson, *et al.*,[1] "Operation of command and control systems is characterized by an enormous quantity of diverse but interrelated tasks—generally arising in real time—which are best performed by automatic data-processing equipment, and are most effectively controlled in a fully integrated central data-processing facility. The data-processing functions alluded to are those typical of data processing, plus special functions associated with servicing displays, responding to manual insertion (through consoles) of data, and dealing with communications facilities."

These anticipated applications seem definitely to discourage a mode of operation wherein a "main program" and its auxiliary routines are conglomerated, by hand or automatically, to constitute a "program deck" which, for each "run," is loaded into the computer and given control. The design of the D825 and its intended uses seem rather to indicate a mode of operation in which many tasks are performed

concurrently, and in which assignments are determined automatically. In such an environment, the operating system must have access to a very large set of programs and data which can be referenced without human intervention. Thus, we have assumed that there is a file of programs and data, which contains not only the raw items, but a substantial amount of information about their nature, interrelationships, requirements, and constraints. The greater part of this information can be collected and appended to the files by the programming system, primarily the compiler in the case of programs, explicitly by the operator, or implicitly by an external signal, in the case of run requests.

RELATED HARDWARE FEATURES

Before discussing the AOSP itself, it will be well to describe briefly the D825 hardware features most closely affecting the structure of the AOSP. These include the switching interlock, the modes of operation of the processors, the system interrupts, and the methods of initiating I/O data transfers.

Switching Interlock

The switching interlock, physically distributed among system modules, provides the interconnections which allow the memory modules to interchange information with I/O modules and processors. The switching interlock consists of a crosspoint switching matrix, which performs the actual switching, and a bus allocator, which detects, defines, and resolves all time conflicts resulting from simultaneous requests by processors and I/O busses for access to the same memory module. Modules simultaneously requesting access to the same memory module are serviced in a bus-dependent order; that is, if the memory module is available, requests from I/O modules pre-empt processor requests. Each requesting module is suspended until access to the addressed memory is attained. If access is not obtained within a specified time, appropriate interrupts occur at the requesting module. (Analyses of queueing probabilities have shown that queues longer than one are unlikely.) The switching interlock relieves the AOSP of module intercommunication scheduling, and also provides indications of possible hardware malfunctions within the system.

Interrupts and Operating Modes

D825 arithmetic/control processors have two modes of operation, normal (interruptible) and control (noninterruptible). Several special control instructions are permitted in control mode in addition to the normal complement of instructions—for example, instructions for setting special registers and transmitting I/O descriptors. In general, operational programs are executed in the normal mode, and control programs such as the AOSP are performed (at least in part) in the control mode. The control mode provides the facility to accomplish necessary control functions without interruption. In addition, complete control of all I/O activity is maintained, since it may only be initiated in this mode.

An automatic interrupt capability is an integral function of the D825. The interrupt system, implemented in hardware, provides the means of switching processor operation from the normal mode to the control mode, either by a specific instruction or by the occurrence of some anticipated event.

Each processor contains an interrupt register. When a particular condition has occurred, a bit is set in a specified position of the interrupt register; the setting of this bit causes this processor to be interrupted. There is also a mask register which can inhibit certain classes of interrupts. The mask register, which may be loaded only in the control mode, allows the AOSP to determine which processors are to respond to particular conditions.

An interrupt initiates an entry to the AOSP. When a processor is operating in the normal (interruptible) mode, its interrupt register is scanned by logic circuits at the completion of each instruction. If a bit has been set in the interrupt register, the processor switches immediately to the control mode (noninterruptible). The various control registers are saved (representing the point at which the task was interrupted), the interrupt bit is reset, and control is established at a point, predetermined by the AOSP, corresponding to the type of interrupt. When the new task is completed, an AOSP instruction causes the processor to switch from control to normal mode and to restore the control registers to their prior normal mode settings. In this way, the interrupted program is resumed at the point at which the interrupt

occurred. (It may be, of course, that some other processor has, in the meantime, been assigned to complete the interrupted program, in which case the interrupted processor is assigned another new task.)

There are two classes of interrupts in the D825: internal (or processor-generated) and external. The internal interrupts are:

1. Real-time clock overflow.
2. Attempt to write out of bounds.
3. Illegal instruction.
4. Parity error from memory.
5. No access to memory.
6. Arithmetic overflow.
7. Halt instruction.
8. No presence bit (indirect address).

The external interrupts are:

1. An I/O operation is completed.
2. An external request (16 possible) has been made.
3. Primary power (which had failed) is now restored.
4. Another processor directs that this processor be interrupted.

If reserved control mode functions are attempted in normal mode, an appropriate interrupt occurs. These functions are:

1. Initiate I/O.
2. Establish base of interrupt transfer table.
3. Interrupt a computer.
4. Load interrupt mask register.
5. Load memory bounds registers.

I/O Initiation

An I/O control module is in one of four states:

1. Transferring data.
2. Held (not released, but not transferring data).
3. Released (available to accept an I/O instruction).
4. Not in the system (either off-line or nonexistent).

When a processor causes a descriptor (I/O transfer instruction) to be transmitted from memory to an I/O bus, the descriptor is accepted and serviced by the lowest-numbered I/O control module on the bus which is in the released state. If no I/O control module is available, the processor embarks upon the conditional transfer specified in the instruction which causes transmission.

When an I/O module accepts a descriptor, it returns to memory a copy of the descriptor received. This echo is checked by the processor to ensure that proper operation has been initiated. When an I/O operation is completed (or terminated for some other reason), the I/O module returns to memory a word (result descriptor) which describes the status of the equipment involved and the cause of I/O termination. At the time of I/O termination, an interrupt signal is sent to the processors, one of which is assigned by the AOSP with the responsibility for determining whether or not the resulting I/O status is satisfactory, and so notifying the initiating program.

INTERNAL FEATURES OF THE AOSP

The AOSP is an executive program whose primary function is to control multi-processing in a multicomputer system. Description of its internal features is preceded in the following by an example of its operation.

Example of AOSP Operation (Multiprocessing in a Multicomputer System)

Envision an arbitrary number—let us say four—of programs in core memory which are suspended (having previously been readied by the AOSP). Consider priorities to be as follows:

Program A: 0.25
Program B: 0.50
Program C: 0.75
Program D: 1.00

Also, consider a D825 system consisting of two processors—call them 1 and 2, where processor 2 is established to process I/O complete interrupts. Initially, processor 1 would assign itself to program D, and processor 2 would assign itself to program C (or vice versa, arbitrarily). Let us assume further, that, in program D, the requirement exists that no further processing of this program can be accomplished until new data is brought into the system. For this type of condition, the compiler (or programmer) would have coded the necessary request for I/O such that the request would be followed by an AWAIT macro. ("I can do nothing more until I receive the new data I told you

about.") The processor executing program D would enter the AOSP in response to the I/O request and would do the following, as directed by the AOSP:

1. Initiate the I/O operation and record the name of the program which had requested it.
2. Suspend program D (store its registers in the register image area reserved in the job table), and mark program D as awaiting an I/O operation.
3. Execute the scheduling program to find the program of highest priority that is capable of being executed (program B, in this case).
4. Load the register image of program B into its registers.
5. Transfer control to program B by performing an interrupt return instruction.

When the I/O operation for program D is completed, processor 2 would be interrupted (since it had been assigned to process all I/O completes), and would perform the following events, as controlled by the AOSP:

1. Store its registers in the appropriate job table entry.
2. Determine the originator of the I/O request, and report to that program (D, in this case) that the operation has been completed.
3. Note that the status of program D is changed in the report.
4. Execute the scheduling program to scan the job table for the highest priority job entry that is capable of being executed (program D, in this case).
5. Load the register image of program D into its registers.
6. Transfer control to program D by performing an interrupt return instruction.

As may be seen in the foregoing example, a program is executed by any free processor, and processors assume control of any free program. If programs C and D were independent subtasks of a larger program (let us say E), the initial assignment of processor 1 to E_1 and of processor 2 to E_2 would involve parallel computation within a single program. This is an inherent facility of the AOSP.

Parallel Control Logic

There may seem to be some difficulty in preventing control interference in a multicomputer system. For example, it might seem that it could conceivably require more program control for a processor to establish exclusive use of some system configuration table than would be required to do the table updating itself. Or, having obtained information from one table, it may be necessary to obtain consistent corresponding information from another table (that, conceivably, another processor is modifying). There are several techniques which could be used to avoid these conflicts. Logic or program controls could be designed to ensure that only one processor at a time can execute the control program, inhibiting (by means of some sort of switch) any other processor from entering the program. Another method might be to allow only some preferred processor to perform control functions; whenever another processor has a control function to be performed, it would interrupt the "control" processor, and have its request processed indirectly.

In the design of the D825, it was determined that, to enhance interrupt response time, and to maintain a balance of interrupt service among the processors, the AOSP may be executed in parallel by any number of processors. It was also determined that some sort of inhibit function must be included in the AOSP to prevent processors from interfering with each other in modifying system tables (such as memory maps or job tables). Associated with each set of modifiable tables and functions is a Boolean variable which indicates whether or not the tables are being modified. This variable may be tested and set with an AOSP instruction. Any number of processors may simultaneously execute the AOSP, each invoking programmed lockouts when use of modifiable tables is required. Certain quantities (subroutine return points, request path indicators, and so forth) do not warrant the use of a lockout. If lockouts were to be provided in these cases, major sections of the AOSP would be executable by only one computer at a time. To avoid this restriction, these quantities are maintained in processor registers, and are request-path-dependent.

Processor Scheduling

After a processor suspends some job, it looks for work. The set of current jobs is maintained in a *job table*. Each job entry in the job table contains the values of the registers of the processor which last suspended the job and a status indication of the reason for suspension.

Typical status indicators are: now running, waiting completion of I/O, available to be run, awaiting some indicator, and so forth. The processor looking for a new job establishes exclusive use of the job table and selects the highest priority job entry which is ready to be run. The processor loads its registers from the register image area of the entry selected, and proceeds to execute the next instruction of the previously suspended job.

A program may have more than one entry in the job table. This can be initiated by a BRANCH macro which causes the generation of another job table entry. In this manner, a program may initiate parallel processing on (potentially) several processors. Of course, if only one processor is in the system, the two paths will be executed in essentially a serial manner. There are advantages accrued in branching, even if only one processor is available. The major use of branching is to conveniently buffer I/O operations and computation. If several independent tape or drum sorts are required, it is a relatively simple matter to separate the functions and allow them to proceed in parallel.

Memory Scheduling (Allocation)

The use of a modular memory configuration introduces slight complications in memory allocation. The memory will not necessarily consist of a consecutive set of addressable quantities, since some memory module may be offline. The *memory map*, composed of an *available space map*, an *in-use map*, and an *unavailable space map*, allows the control program (in the AOSP) to determine which areas are in use and which areas are available. To increase system efficiency or to provide some sort of superglobal area, a particular block of memory may be reserved for whatever use a particular installation requires. The AOSP is protected in this way.

When a processor must allocate space for some requestor, it attains exclusive use of the available space map. The smallest adequate block is selected and removed from the available space map. If adequate space is not currently available, a deferred space request entry is established, which will be serviced whenever the required space becomes available. The deferred space request item indicates the amount of space required, the name (if any) of the block, the type of object this will later represent, and identification of the requestor(s). An attempt is made to allocate a deferred request whenever the size of the largest block of available space increases. The priority of the deferred request increases with time, and if the priority reaches a certain threshold, allocation of any request is suspended until this deferred request is serviced.

USER FACILITIES OF THE AOSP

Some of the many user facilities provided by the AOSP are discussed in the following. (Some of the more important details of program format are discussed in Appendix I.)

Initiation of Jobs

The AOSP reacts to all external request interrupts by scheduling the execution of programs which the user has designated to interpret messages from external lines. The action to be taken upon receipt of a message on the external lines depends upon the inherent priority of the requesting device and the content of the message. Associated with each external request line is the corresponding decoding program, an area in memory in which to place the message, and the inherent priority of the requesting line. An external interrupt indicating the presence of a command message is the initiating signal for a new process (called a *job*). If the message is a run request (a new process to be initiated) the required program (or programs) is fetched from the files, parameters are established and control is given to it. Note that there may be several run requests for execution of the same program on different data sets. Hence, requests, as well as programs, are given identification.

I/O Facilities

Some of the features of the AOSP facilitate multiprocessing. Several independent user programs may reference a single shared I/O device (output tape, library tape, etc.), and facilities are provided to prevent interleaved use by two programs of serial-read/write devices (card punch, reader, printer). To prevent two programs (being run by two processors) from alternately reading small consecutive pieces of data from mutually remote areas of the same tape, one (or both) of the programs must first attain exclusive use of the device to prevent

excessive tape motion. Different rules apply if shared input and output tapes are used, with subsequent processing, than if real time functions are dominant.

Associated with each I/O device is a table which indicates current allowable usage of this device. In each such table is a *status controller* which indicates the name of the job which controls how the device may be used. Only the status controller can establish how this piece of equipment may be used (read only, write only, allows no other user, allows any user, allows any job to attain exclusive use, etc.). In addition to the status controller, the current exclusive user, if any, is maintained. Tape position and reel numbers are maintained. Programs may make requests of the type: "Read the 14th record after the fourth end-of-file mark on tape reel #143"; and "Establish this job as status controller of the tape unit on which reel #236 is mounted."

If an I/O request cannot be honored, the requestor is terminated. This procedure could be replaced with other alternatives. The job could be given an error answer. A message could be printed requesting the operator to provide the required equipment, with the job suspended until he has done so. Our current thinking is that a job scheduling routine should not activate a job until its I/O requirements are available; hence, termination is a reasonable action.

File Facilities

A *file* is a set of named objects which, in some sense, logically (but not necessarily physically) belong together. At the base of the AOSP filing system is a *system directory* table which contains the name, size and location of the *file directory* for each file in the system (whether active or inactive). Each file directory is of similar format and contains the name, size, and location of every object in the file. The directory of a file need not be stored on the same device as the objects in the file.

Additional AOSP Facilities

The AOSP also provides facilities for user programs to make run-time requests for named data structures, procedures, and I/O devices. One of the major functions of the AOSP is the coordination of I/O activity. Buffered I/O transfers are provided for the user as a matter of course. In addition to a rich vocabulary of I/O requests, facilities are provided to allow run-time priority modification, initiation of a parallel process, establishment of set-up parameters for a procedure, and file modification during servicing of requests from a file. There are also specialized time-dependent provisions, such as "Suspend this job for n milliseconds of real time." (A real-time clock is included in the D825.)

Anticipated Extensions

We anticipate that, after further study and development of the AOSP, run requests will be filed. A job scheduling program will be activated at appropriate times to introduce new work for the system by analyzing the requirements of runs (such as time, storage space, branches, priority, and so forth) and the environment of the system. In this way, more efficient control can be maintained of equipment usage.

SUMMARY

The paper describes the initial implementation of an operating system for a multicomputer data-processing system. Major emphasis in the operating system design has been placed upon coordinating several processors sharing memory and I/O equipment while executing many concurrent tasks. Important features of the program include the response to interrupts, run-time allocation of memory, coordination of I/O activity, initiation and removal of programs, and filing facilities.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the outstanding efforts of their many colleagues at Burroughs Laboratories who have contributed so well in the design and implementation of the D825 Modular Data Processing System and the Automatic Operating and Scheduling Program. In particular, we wish to acknowledge J. Anderson, W. Hopper, and S. Hoffman for the initial system concept; L. Mott and the logic design staff; and J. Maestrelli, F. Zurcher, A. Fox, and E. Amarnick for the implementation of the AOSP.

The authors also wish to acknowledge the efforts of K. Sattley and S. Warshall of Computer Associates Inc., for their contributions to the concepts and initial design of the AOSP.

REFERENCE

1. ANDERSON, JAMES P., SAMUEL A. HOFFMAN, JOSEPH SHIFMAN, and ROBERT J. WILLIAMS, III, "D825—A Multiple-Computer System for Command and Control," *Proc. FJCC*, 1962, p. 86.

APPENDIX I—PROGRAM FORMATS

The basic program entity in memory consists of two parts: the body of instructions, called the *program area* (PA), and a block of words used as data by the program, called the *data area* (DA). (Several DA's may, of course, correspond to a single PA, since any program may be executed simultaneously by two processors without mutual interference.) The PA and DA blocks correspond to the two principal base registers in the processor—the *base program register* (BPR) and the *base address register* (BAR). The BAR and BPR control relative addressing. When a processor is executing a program, the BPR of that processor is set to the beginning of the PA, and the BAR normally contains some address within the DA of the program. When a program is run simultaneously by two processors, the BPRs of the two computers are set to the same value, but in general, the BARs are set to different values. Data objects allocated independently of the DA of a program which refers to them are termed *external.*

To meet the bookkeeping requirements of the AOSP, both the PA and DA are preceded by a few additional words of information. The words preceding the PA are called the *program header* (PH), and those preceding the DA, the *data header* (DH).

Examples of the format of program headers and data headers are illustrated in Figs. 2 and 3. The first five lines of the headers are created and maintained by the AOSP. The remaining lines of the headers, the *adaptor blocks,* are created at compile time, and remain in this form in the system files.

The adaptor block contains the names of data or program items which, for various reasons, cannot conveniently be allocated within the prime program areas or data areas. Whenever a program is made ready for execution, the external data objects referred to by the program are obtained by the AOSP, and the current addresses of these objects are inserted in the appropriate locations within the adaptor block of the DA.

As shown in Figs. 2 and 3, one area in the adaptor blocks of the program area is the *names of necessaries.* Also shown, in the adaptor block

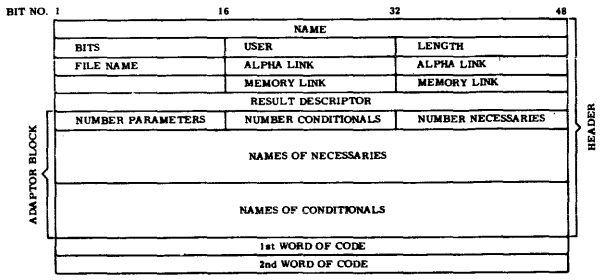


Figure 2. Program Header.

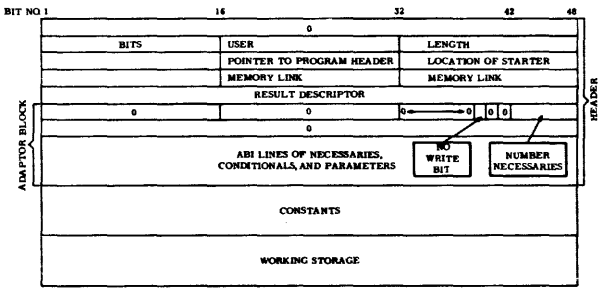


Figure 3. Data Header (Before Ready)

of the data area, are the *ABI lines of necessaries.* ABI stands for the "Ith adaptor block line." One of the functions of the AOSP is to ensure that all of the "necessary" items required by the program are in core memory at run time. (Necessaries may be library subroutines, data objects, or other programs in the files.) Let us suppose that the necessary item for a particular program is an array or table of data, and that this table has the name HENRY. The program would then have a necessary line for this data object in both the data header and the program header. The two necessary lines in question would appear as in Fig. 4.

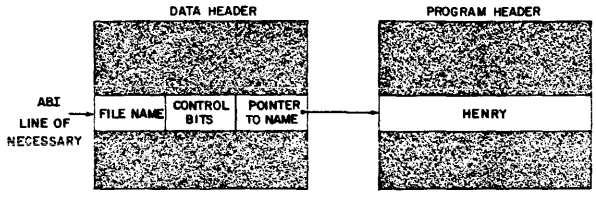


Figure 4. PA-DA ABI Linkage (Before Ready)

In preparing this program for execution, the AOSP would first determine whether there were any external blocks necessary for the execution of the program. In this particular case, the AOSP would scan the program header and data header and find the requirement for the data object HENRY. The AOSP would then initiate a routine that would first scan the list of data blocks presently in core memory. If HENRY were in core memory, the pointer in the adaptor block of the DA would be changed to the absolute location of the data object, and the file name would be changed to the link of users of HENRY; the presence bit would be set to permit indirect addressing by the object program.

If HENRY were not in core memory, the AOSP would initiate a chain of operations that would locate HENRY in the system files, allocate space in core memory for HENRY, and bring HENRY into the allocated core space. The address field in the adaptor block would then be changed to the absolute address of HENRY.

It should be noted that this sequence describes the handling of one necessary item. In general, there may be many necessary items, and it is the function of the AOSP to see that all are in core memory at program run time. Indeed, some of the necessary items of a program may themselves have necessary items, and the AOSP would fulfill the same function for these. Only when the whole structure is in core memory will the AOSP report to the original requestor that the program is ready to be run.

Some items required by a program may be of a conditional nature. This is especially true where procedures or data sets of programs are required only as the result of a particular decision at run time. It would be wasteful to clutter the core memory with these items if they are not to be used in the normal program cycling. This type of item is considered *conditional.* Adaptor block entries of conditional items are also carried in the program and data headers, but are not readied initially. The readying of conditional items is identical to the readying of necessary items, but the items are only readied as a result of macro calls on the AOSP by the program during the course of the program execution.

Table I. Features of D825 Hardware

Memory module:	16 modules maximum
	word length: 48 bits plus parity bit
	4.33-microsecond cycle time; 1-microsecond access time
	4096 words/module
	5 busses/module
Arithmetic/control processor (computer module):	3 Mc logic rate
	normal and control modes of operation
	program-controllable interrupt system
	128 magnetic thin-film storage registers (300 ns cycle time)
	four-level operand stack (thin films)
	independent program and data base address registers
	0-, 1-, 2-, and 3-address instructions
	15 index registers
	syllable-string instruction format
	one to seven syllables per instruction
	n-level indirect addressing
	data presence bit test
	fixed, floating, and field instructions
I/O control module:	10 modules per exchange, maximum
	500 kc maximum character (six bits plus parity) rate in I/O exchange
	32 input channels
	32 output channels
	transmits I/O complete interrupt signal to every processor

A TIME-SHARING DEBUGGING SYSTEM FOR A SMALL COMPUTER

J. McCarthy
Computation Center, Stanford University, Stanford, California
S. Boilen
Bolt Beranek and Newman Inc., Cambridge, Mass.
E. Fredkin
Information International Inc., Maynard, Mass.
J. C. R. Licklider
Advanced Research Projects Agency, Department of Defense

The purpose of the BBN time-sharing system is to increase the effectiveness of the PDP-1 computer for those applications involving man-machine interaction by allowing each of the five users, each at his own typewriter to interact with the computer just as if he had a computer all to himself. The effectiveness of this interaction is further enhanced by the use of the TYC language for controlling the operation and modification of programs.

First the computer. The PDP-1* is a single address binary computer with an 18 bit word and five microsecond memory cycle; most instructions require ten microseconds to execute. The basic memory size is 4096 words, but up to 65,536 words may be addressed indirectly. The machine we used has 8192 words, 4096 of which are reserved for the time-sharing system. Each user sees a 4096 word memory. We shall describe further relevant features of the computer later.

* The PDP-1 computer is manufactured by Digital Equipment Corporation of Maynard, Massachusetts. All the equipment described in this paper was made by D.E.C. and their cooperation in developing and building the modifications and additions to the basic computer required for time-sharing was essential to the success of the project.

Attached to the computer is a high speed magnetic drum memory divided into 22 fields each of 4096 words. A basic operation of the drum system is the *memory-swap* accomplished in 33 milliseconds. In this operation 4096 words are transferred from the core memory to a drum field and simultaneously the core memory is loaded from a different drum field. This permits the following time-sharing mode of operation.

A 4096 word drum field is allocated for saving the *core image* of each user when his program is not running. A user's program in *run status* is run for 140 milliseconds, then if there is another user also in run status, the state of core memory is stored in the first user's core image on drum and simultaneously the second user's core image is loaded into core and the second user's program is started in the appropriate place. In the worst case of all five users in run status, the system makes its rounds in $5 \times (140 + 33) = 865$ milliseconds. For man-machine interaction this means that if the user types a character calling for a response from his program that requires less than 140 milliseconds (= 14,000 machine instructions), he will get the first character of the response in less than .865 seconds. This worst case is ex-

pected to be rare because when a user's program types out a multicharacter message, successive characters go into a buffer area in the system core; when the buffer is full the user's program is removed from run status until the buffer is nearly empty. Therefore, users with extensive output spend very little time in run status and the other users get correspondingly quicker service. (In fact the condition in which no users are in run status is expected to be so common that a provision for a background program to be run only when no time-sharer is in run status is contemplated.)

THE TYC CONTROL LANGUAGE

The language used to control the debugging is adapted from the DDT language devised by the TX-0 and PDP-1 group at M.I.T. directed by Jack Dennis for the TX-0 and PDP-1 computers. The use of typewriters rather than the console switches for on-line debugging has been developed at M.I.T. and M.I.T. Lincoln Laboratory since 1957, and at BBN since 1961. These languages have greatly increased the effectiveness of the TX-0, TX-2 and PDP-1 computers and are now being developed for the IBM 7090 time-sharing system by F. J. Corbato of the M.I.T. Computation Center. Unfortunately, except for a recent paper by Corbato, the work has not been published. Our only bow at history is to credit John Gilmore with the first such system for the TX-0 in 1957.

First, we shall consider the facilities for examining and changing registers. Suppose the user wants to examine register 344. He types

344/

and the computer types back the contents of this register interpreted as a computer instruction if possible. Thus it might type back "add 4072". The user then has several options.

1. He can carriage return and close the register if he is satisfied with its contents.

2. He can ask to see the next register by hitting the backspace key. The computer might then type back

345/ sub 4075.

3. He can ask to see the previous register.

4. He can ask to see register 4072 by hitting the tab key.

5. He can ask for the contents of the register as an octal or decimal number.

6. He can change the contents of the register by typing new contents such as "add 4073" and then hitting one of the delimiter characters.

The computer user who does not have experience with systems of this kind may not immediately realize the increase in effectiveness that these facilities alone give over console debugging. The advantages are that typing is easier than flipping switches, that a record is obtained of what was done, and that useful features can be added to the control language.

As a further feature the user can define symbolic names of addresses so that his typeouts can take the form

a/ add b + 3
a + 1/ sub b + 6

In order to start a program say at register 3145 the user types 3145G. This gives the typewriter up to the program, but he can get it back for control purposes by typing *center dot.* If he does so he can interrogate and change registers without stopping his program but if he wants to stop it he types H. Once it is stopped he can start it where it left off by typing G without a numerical argument. He can also interrogate the arithmetic registers either while the program is running or while it is stopped. Of course, the contents of a register interrogated when the program is interrupted at a random moment may not be significant; this depends on the program.

Except at the beginning and the end of his session the user does not ordinarily use the paper tape apparatus. Instead he designates a position on the drum for the punch and a position for the reader using TYC. An instruction in the user's program to punch a character onto paper tape actually results in entering the character into a buffer for transmission to the drum when the buffer is full or no punch instructions have been given for a while. This feature allowed us to make available in the time-sharing system symbolic assembly programs and other utility programs that were developed previously.

The TYC language is described in detail in reference 7.

RELEVANT FEATURES OF THE PDP-1 COMPUTER

In order to explain the detailed functioning of the BBN time-sharing system, it is necessary to understand the input-output system of the PDP-1 computer, the sequence break system and the restricted mode.

1. Typewriters and paper tape

Each *iot* instruction transfers a single character between the 18 bit *io* register of the computer and the external device. All *io* instructions have the same left 5 bits, the 6th bit is normally used to determine whether the wait before the next instruction is determined by the external device or whether the program keeps this responsibility. The remaining 12 bits of the instruction are used to determine the device and the direction of transfer. The characters are 8 bits on paper tape and 6 bits for typewriters. When the computer is not in sequence break mode a character typed by the user results in turning on a sense flag which can be interrogated by the program to determine when to pick up the character.

2. Drum

In transfers from the drum the number of words to be transferred and the locations in core and on the drum are specified. During the transfer the arithmetic unit of the machine is tied up.

3. The machine has other input-output devices but their operation does not have to be described in this paper.

4. The sequence-break system.

Besides the simple form of input-output described above, the machine has a sequence-break system which permits input-output and computation to be carried out simultaneously and asynchronously. The BBN time-sharing system makes full use of sequence break.

The sequence-break system has 16 channels to the outside world arranged in a priority chain with channel 0 having highest priority and channel 17_8 lowest priority. Associated with each channel are four registers in core 0. When a signal comes from an *io* device that the device has a character ready for the computer or is ready to receive a character from the computer, an interrupt will occur if an interrupt is presently allowed on that channel. When an interrupt occurs the contents of the *ac* (accumulator) *io* (input-output) and *pc* (program counter) registers are stored in three of the registers and control is transferred to the fourth which must therefore contain a jump to a program for dealing with the interrupt. When this program has finished its work it must execute an indirect jump through the register where the program counter was stored when the break started. This returns control to the program that was interrupted and tells the sequence break system that the break is over. Once a break is started on a channel a break cannot occur on a lower priority channel nor can another break occur on the same channel until the break is over. Each channel can store one break until it is allowed to occur.

5. Restricted Mode

This mode was devised specifically for the time-sharing system. When the computer is in this mode and if there is no break started in the sequence break system, then any of the following events lead to a sequence break on channel 16.

1. An attempt to obey an *io* instruction.
2. An instruction that would normally stop the computer such as a halt instruction or an illegal instruction.
3. An attempt to refer to core 0.

The instructions that enter and leave extend mode and restricted mode are considered *io* instructions. The time-sharing system operates only when a sequence break has occurred and hence is not subject to the restrictions.

6. The Channel 17 Clock.

Every 20 milliseconds a signal for a sequence break on channel 17 occurs. Programs can turn off channel 17 (or any other channel) by an *io* instruction if they don't want breaks to occur. Note that channel 17 is the lowest priority channel. The clock is a multivibrator whose period is controlled by a potentiometer.

HOW THE TIME-SHARING WORKS

The time-sharing executive program is not readily described by a single flow chart because its different parts act asynchronously as determined by sequence breaks. It includes the following programs:

1. The typewriter *io* program

Associated with each typewriter is an input-output program and a buffer area. These pro-

grams are entered when sequence breaks occur. Suppose a user types a character w. The program knows whether the character is addressed to the user's program or to TYC.

On the other hand, if the interrupt comes from the completion of the type-out of a character the program types the next character from the buffer if any.

After transferring the information the break is ended.

Channels 6, 7, 11, 14 and 15 are allocated to typewriters.

2. The channel 16 dispatcher.

The computer is in restricted mode during the operation of the time-sharing system. As we stated earlier, this means that *io* instructions and instructions that halt the machine lead to sequence breaks on channel 16. The user programs his input-output just as if there were no time-sharing system. Therefore, when a channel 16 break occurs the program first looks at the instruction that caused the break. Suppose the instruction is a type-out instruction. If the type-out buffer is not full the character that the user program wants to type is added to the buffer and if necessary a sequence break on the typewriter channel is instigated to start typing. If the type-out buffer is full the program must be dismissed from run status. If the instruction is a type-in instruction, a character is given to the user program if there is one in the buffer; otherwise the program must be dismissed from run status.

If the instruction is discovered to be one that halts the machine, the program is dismissed from run status and a note is left for TYC to tell the user what happened.

Paper tape input and output is handled in a similar way except that the dispatcher must check whether the user has the punch or reader assigned to him. If not, the user's program is dismissed, and a complaint is made to him. As soon as the reader or punch has been relinquished he can continue the program from where it left off.

The typewriter and paper tape instructions are interpreted and simulated by the channel 16 dispatcher so precisely that programs written before the time-sharing system was developed can be run without change in the system, provided they do not themselves use the sequence break system. This means that almost all the previously used symbolic assemblers, typewriter input-output routines, text editors etc. can be used without change and that the user can use the TYC language to debug routines that are to be used outside it.

3. The channel 17 clock routine.

Every 20 milliseconds or so a sequence break signal is given on channel 17. Since channel 17 is the lowest priority channel this break can take effect only when no typewriter, paper tape or channel 16 dispatcher break is in progress. Moreover, except when the channel 17 program turns off the sequence break system it can be interrupted by typewriter or paper tape sequence breaks.

The basic task of the channel 17 clock routine is to decide whether to remove the current user from core and if so to decide which user program to swap in as he goes out. A user may be removed from core for any of several reasons.

1. His quantum of time is up and he should be put on the tail of the queue.
2. He has filled an output buffer.
3. He has asked for a character and there is none in the input buffer.
4. He has tried to execute an illegal instruction or to use input-output equipment not available to him.
5. The typewriter control program has filled a buffer or has finished a request concerning his program.

If the channel 17 routine decides to remove the current user it will swap into core the next user in the round robin who is in run status. A user not in run status can become so for any of the following reasons.

1. An output buffer is almost empty.
2. A character requested by his program has arrived.
3. The typewriter control program wants him in core to interrogate registers, to change them, or to run the program.
4. The typewriter control program (TYC).

The typewriter control program is in core 0 and it interprets and obeys requests from the user to give him information about his program, to change it, to run it, and to stop it. The same program must work for all users and whenever a user is put in core TYC is modified so that it refers to the current user's program.

The user makes his requests and receives information from TYC using the same typewriter as his program uses for input and output. Therefore, it is important that no matter what program the user is attempting to debug he shall be able to regain control if it goes astray. This is accomplished as follows: When the user starts a program running he can either retain the typewriter for control purposes or else give it up to the program. If he gives the typewriter to the program, then characters it types appear on his typewriter and characters he types are given to his program if it asks for them. Suppose his bad program is taking characters from the typewriter but ignoring them. He can then type the character center dot "·" which is a non-spacing character on the PDP-1. If he follows this by a carriage return the typewriter is then in the control of TYC and subsequent characters are interpreted by TYC. If he actually wants *center dot* to be transmitted to his program, he must type it twice.

Suppose now that his program is in an output loop and refuses to stop typing. Then he turns the power off on the electric typewriter. The result of this is that the computer fails to get a "done pulse" from the next character typed within a second. Control then goes to TYC which tells the channel 17 program to dismiss the user's program from core and returns the typewriter to the control of TYC as soon as the power switch is turned on again.

APPLICATIONS

The most obvious application of the BBN Time-Sharing system is to speed up debugging by allowing each user more console time and good debugging languages. In our opinion the reduction in debugging time made possible by good typewriter debugging languages and adequate access to the machine is comparable to that provided by the use of ALGOL type languages for numerical calculation. Naturally, one would like to have both but this has not yet been accomplished on any machine.

We can now mention some other applications that our system makes possible by providing inexpensive console time.

1. *Small calculations.* At present there is a large gulf between desk calculators and computers. One can start getting results 10 seconds after sitting down at a desk calculator, but extensive calculations are very tedious. The BBN Time-Sharing System makes possible and economically reasonable providing a continuous transition from using the computer as a desk calculator at one extreme to writing ALGOL programs at the other. An intermediate step is a system that allows the user to define functions by statements like

$$f(x) = x \uparrow 2 + 3.0x \times + 4.3$$

or even

$g(m,n)$ = if $m>n$ then $g(n,m)$ else if $rem\ (n,m) = 0$ then m else $g(rem\ (n,m), m)$

and then be able to use these functions in arithmetic calculations by writing something like

$$g(3,21) \times f(38) + 19 =$$

and have the computer print the answer by interpreting the formulas for the functions. To some extent this has been achieved by the program "Expensive Desk Calculator" written by Robert Wagner at M.I.T.

2. *Editing Texts.* Several programs have been written to use the PDP-1 computer to edit paper tape texts. The user can originate text, make insertions and deletions, display the corrected text to make sure it is correct, find all occurrences of certain strings of symbols. These editing programs are much more convenient for correcting programs than using flexowriters or than making changes in a card

One such program, Colossal Typewriter, operates as follows:

There are two modes, text mode and control mode. When the program is in text mode, each character typed by the user is added to a buffer held in the core memory of the computer. There are four exceptions to this: If the user types a backspace, the program deletes the last character from the buffer, and this operation can be repeated as many times as may be required to correct a local error. Another character causes the program to type out the last 20 characters in the buffer, and a third returns the computer to control mode. The fourth special character—the single quote ‘ causes the cliche whose name is the following character to be entered in the text if there is such a cliche. Thus typing ‘ a causes the cliche named a to be entered. In addition, the cliche Feature may be used to enter any of the four control characters into the text. The user need only type ’ followed by the character in question.

In control mode the user has the following facilities: to type out the buffer, to punch (pseudo-punch) the buffer, to read (pseudo-read from the drum) into the buffer a number of lines or until the first occurrence of a given cliche, to reset the ends of the buffer, to give the current buffer a name as a cliche, to kill the buffer, and to go into text mode.

Other text-editing programs allow the use of the CRT on the computer to display lines of text.

3. *Teaching Programs.* An experimental teaching laboratory using a system based on the principles of this paper is being installed at Stanford University.

Additional applications of large time-sharing systems are described in (2), (3) and (5).

OPERATING EXPERIENCE

The BBN Time-Sharing System has been in operation at Bolt Beranek and Newman Inc. since September 1962. The computer is operated in the time-sharing mode four hours per day. Initially, the number of typewriters was two but this has been increased to five. The present system has been found to have the following weaknesses which we hope will be corrected: There is no program library on the drum so that excessive use of the paper tape reader is required. Magnetic tape files for user programs are not available in the system. Five computer operated typewriters in one room make too much noise. Versions of the utility programs especially adapted to time-sharing are desired.

EXTENSION TO LARGER COMPUTERS

It is worthwhile to ask to what extent the time-sharing technique described in this paper is of more general use. As a computer the PDP-1 is characterized by high speed and relatively small memory. Its low cost means that it will not ordinarily have to be shared by a very large number of users. Suppose we wanted a time-sharing system based on core-drum swapping on another computer. Suppose that

n = number of simultaneous users
t = time for a memory cycle
m = number of words in user's memory that have to be swapped
r = response time
f = fraction of time taken by swaps;

then we have

$r = n\,t\,m(1 + \frac{1}{f})$ under the assumption that the drum keeps up with the core memory and that the read and write halves of the core memory cycle are used separately.

In the present case if we put $f = .25$, $n = 5$, $t = 5 \times 10^{-6}$ sec $m = 4000$ we get $r = .5$ sec. The difference between this result and the actual maximum response time of .85 sec. comes mainly from the fact that the present drum system swaps a word about every 8 microseconds instead of every five microseconds which in turn comes from using a standard 1800 rpm motor on a drum on which each track has 4096 bits around.

If we were to make a similar system for the IBM 7090 computer, we could have $n = 5$, $t = 2 \times 10^{-6}$, $m = 16{,}000$ (the memory of this machine really consists of 16384 72 bit words) $f = .25$ and would get

$r = .75$ sec

Thus, on account of its much larger memory the 7090 would have about the same relation between number of users and maximum response time as the PDP-1. This is less satisfactory because the expense of the larger machine requires it to serve many more users. Nevertheless, such a system would still be useful. If we make the more optimistic but reasonable assumptions that only $\frac{1}{5}$ of the users sitting at typewriters will be in run status at a given time and that a 3 second response time is tolerable, then the system could accommodate 100 typewriters which is economically quite reasonable. This would require a better drum system than is available connected so as to allow memory swaps at core speed.

REFERENCES

1. J. GILMORE—Lincoln Lab memo (out of print)
2. C. STRACHEY—Time-Sharing in Large Fast Computers in Proceedings of the International Conference on Information Processing, UNESCO, Paris 15-20 June 1959, UNESCO, Paris, 1960, pp. 336-341.
3. J. C. R. LICKLIDER—"Man Computer Symbiosis". *IRE Transactions on Human Factors In Electronics,* (March 1960).

4. F. J. CORBATO—1962 WJCC An Experimental Time-Sharing System, Fernando J. Corbato, Majorie Merwin-Daggett, Robert C. Daley, 1962 Spring Joint Computer Conference.
5. J. MCCARTHY—Time Sharing Computer Systems in Management and the Computer of the Future edited by Martin Greenberger, M.I.T. Press, 1962.
6. PDP-1 manual—Digital Equipment Corporation, Maynard, Mass.
7. S. BOILEN—User's Manual for the BBN Time-Sharing System—Bolt Beranek and Newman, 50 Moulton St., Cambridge, Mass.

THE ATLAS SCHEDULING SYSTEM

D. J. Howarth, P. D. Jones and M. T. Wyld

INTRODUCTION

Atlas is the name given to a comprehensive computer system which has its origin in the Computing Machine Laboratory under Professor T. Kilburn at Manchester University. The final design is the result of close collaboration between this group and Ferranti Ltd. This paper describes the basic principles of the scheduling system adopted for Atlas. The main features discussed are the ordering of a queue of jobs awaiting execution, the allocation of main store space and tape decks, the assembly and preparation of jobs for execution and the communication system between operator and scheduler program, which includes the allocation of external priorities to jobs. A later paper will describe the dynamic time-sharing of jobs during the course of their execution, and associated simulation studies.

Virtually no information has been published which is of practical value in formulating the Atlas scheduling system, and hence the system is designed to facilitate alterations in the light of operating experience. The basis of the system may be expected to require little changing, but certain decisions, which depend upon a balance of various factors, may require modification when the relative importance of these factors is known in more detail.

SUMMARY OF THE ATLAS OPERATING SYSTEM

The principles of the Atlas Operating System, all the activities of which are controlled by a program called the supervisor, have been described in previous papers.[1, 2, 3] In order to appreciate the action of the scheduler, it is advantageous to summarise first the salient features of the system. It is designed to overlap the operation of those parts of the computing system which can function concurrently, namely, input and output peripherals, magnetic tapes, the central computer and the human operators. In order to achieve overlap of input and output operations with computing, regions of the core and drum store are used as "wells," which can be filled and emptied at peripheral speeds by the peripheral equipment, and at computer speeds, for short bursts, by the central computer. These wells are supplemented by system magnetic tapes (a system magnetic tape is one which is controlled by the supervisor in contrast to a private magnetic tape which is under direct control of an ordinary program), which are not used to supply the central computer directly, but which are used to fill and empty the wells in core and drum store. Jobs whose peripheral input is complete are assembled in main store, prior to execution, from the system input tape or from other system tapes on which they have been loaded previously. Program results occupy the output well in main store before being recorded on the system output tape.

Complete jobs may consist of several separate input documents (a document is a self-contained section of input information presented to the computer consecutively through one input channel), one of which must contain a job description. This lists all other input documents required, the titles of any magnetic tapes required, the output stream and the type of peripheral required for each, and also sup-

plies approximate estimates of storage space (in combined drum and core store) required for execution, computing time, and quantity of output for each output stream. These estimates are used by the supervisor program as a guard against serious program error and are also available for use by the scheduler. Decisions called for by the operator are thereby reduced to a minimum, sufficient information being supplied to the computer by the user to enable a job to be run. Allowance is made for operator intervention in exceptional cases and this is described in a later section.

The scheduler is the part of the supervisor program which determines the order in which the available jobs awaiting execution should be assembled in main store, and should be compiled and executed. In the following sections it will be assumed that no special priority has been attached to jobs by the operator, and that the scheduler is permitted to select jobs in any order; we shall consider later how this order may be influenced by the action of operators.

THE ADVANTAGE OF A COMPUTER ORGANISED SCHEDULING SYSTEM

After recognising that the proposed dynamic buffering scheme (described above) on Atlas means that normally all slow peripheral transfers are indirect and overlapped with computing operations, the merit of doing anything other than executing jobs in precisely the same order as they enter the computer might be open to doubt. However, this simple method of operation can lead to extremely inefficient use of the computer. This would be the case, for example, if a long sequence of jobs all produced output for one peripheral, while other peripherals, which later jobs wished to use, remained idle. On a very large computing system (32 tape decks, 4 line printers, etc.) the order in which jobs are executed is not so important since there may be enough peripherals to deal with any load, irrespective of the order in which jobs are executed. However, it is uneconomical to buy a computing system with more than the minimum number and types of peripherals, which, if used efficiently, can handle the expected load.

If jobs are sorted by the operator before they enter the computer, then inefficiencies of this type are less likely to occur. However, due to the speed at which the machine executes jobs, and also the operator's comparative lack of knowledge of the immediate state of the machine, the task of efficiently ordering a large number of jobs is extremely difficult. A far better method of operation is for the operator to take no account of the types of job entering the computer (all jobs being fed in as soon as they appear), and for a scheduling program, which at any instant knows the exact state of the computer, to determine the order of execution.

A sophisticated and efficient scheduling system is the ultimate component of the overall Atlas supervisor, made possible by the special time sharing features of the Atlas hardware and basic supervisor routines. If a computing system does not have built in features for protecting and interrupting programs during execution, then anything other than a very simple scheduling system is probably not worthwhile, even if it is possible to design some elegant system. In these circumstances the advantages of an "efficient" scheduling system are offset by the time and effort required to devise the system, and then, when the scheduler is working, by the time spent in the scheduling routines.

On Atlas, the main difficulty lies not so much in the implementation of any given scheduling system, but in formulating the particular rules and framework of the system. If the scheduler is designed properly, it should be possible, in the light of operating experience, to tailor it to obtain the best results for the particular installation concerned and type of work it is doing. Excessive scheduling, when considerable time is spent in routines doing detailed look ahead processes, should be avoided: inaccurate estimates and error monitoring of jobs tend to nullify any attempt at detailed future planning.

THE TASK OF THE SCHEDULER

The object of the operating system on Atlas is to maintain the fullest possible useful activity in those parts of the computing system which can function simultaneously; that is, to reduce to a minimum periods of idleness in any part of the system which is required for further use. Such periods can be caused by a delay in passing information from one branch of the system to another; for example, if a job is using a magnetic tape, either the central computer may

be idle awaiting the conclusion of a transfer between store and tape or the tape may be idle awaiting a command from the central computer. A period of idleness can also occur when a peripheral is free and no attempt is being made to execute available jobs which could use this equipment.

The scheduler may be visualised as arranging for the transfer of jobs from a list of available jobs to a list of jobs in course of execution. The supervisor program on Atlas permits sharing of the central computer between several object programs in course of execution, and thus the Execute List may hold more than one job at any one time, if this is necessary to achieve efficiency; the composition of this list will be described in this section. At first sight, it would appear that maximum efficiency would be obtained if the Execute List comprised one job using each output peripheral and magnetic tape, together with a base load job to use any available central computer time. Due to the wide difference in the rate of use of information by the peripherals and the central computer, the central computer might be expected to be in use for only a small proportion of the total time on any one problem, and whilst not in use for this problem could be used by other problems. The presence of so many problems on the Execute List is not necessary to achieve efficiency however, and is to be avoided in the interests of efficient use of store. The operative parts of such problems must be in core store to permit fast switching of control between problems without the need for transfers between core and drum stores; this would imply an excessively large core store, any one part of which is only in use for a small proportion of the total time. Inefficient use of store also results from the fact that, for many problems, the combined core and drum store required for execution considerably exceeds that required for storage of input material, and hence total storage requirements are reduced by reducing the number of problems present at any one time on the Execute List. Since frequent switching of control between object programs is not desirable, the supervisor program is designed to provide rapid switching of control between an object program and supervisor routines, rather than between two object programs.

A straightforward reduction in the length of the Execute List is achieved by using the output well in core and drum store and on the system output tape to collect output information from all object programs for all output peripheral equipments. Problems which are output limited (that is, those which use an output peripheral for a longer time than they use the central computer) can then be executed in series, filling the output well. During their execution, such jobs are not held up for peripheral transfers. Only when the output well is filled for all peripherals is control switched to programs which are computer limited, only one of which need be in course of execution at any one time. The output well is emptied by the output peripherals until a low level of available output is reached for any peripheral, when output-limited jobs are again executed in series to refill the output well. In the absence of jobs using magnetic tape, therefore, the presence of two jobs on the Execute List achieves efficiency, and for most of the time only one of these is actually active. Jobs using magnetic tapes cannot be dealt with in this way, since it is impractical for the supervisor program to provide an adequate buffer for information transferred to and from magnetic tapes. If tape limited problems are run, therefore, it is essential in the interests of efficiency to maintain these on the Execute List in addition to the two entries described above; the central computer will use tape waiting time to proceed with other jobs in the Execute List. The number of such jobs will depend upon their computing time, and the number of tape decks required; one such problem may be sufficient under normal conditions on many Atlas installations.

The object of the scheduler is to maintain a supply of problems to the Execute List and to arrange as far as possible that a vacancy on the Execute List can be filled immediately by a problem already assembled in the input well in main store. The scheduler will therefore be activated whenever a vacancy appears on the Execute List, through termination of a problem, and whenever input of a job through the input peripheral equipments is complete. Since no peripheral limited problem (i.e. problems which are entered to the Execute List in order to supply output to the peripherals) will be executed once the output well is full, and while the peripheral equipments are emptying it, the scheduler is also activated whenever the level of available output for any peripheral which is

in use reaches a lower limit. Problems are then executed in series until the output well is filled, and then a vacancy is left on the Execute List. By this means, frequent interruption of a long computer-limited job by short output-limited jobs is avoided; the latter are run in bursts, during which the computer limited job becomes inactive, and between these "bursts" the computer limited job is free of interruption. Following sections will describe in more detail the types of jobs handled by the scheduler, and it will be found that the simple description so far given of the necessary entries in the Execute List is capable of extension to embrace all jobs which occur in practice.

STREAMS

Having stated the general aims and method of the scheduler there remains the practical problem of how this program selects the right type of job from those available and assembling it for execution. The major difficulties connected with selecting jobs of the right type from a single long list of jobs are the time required for searching and the possibility that some job may be permanently overlooked since it never fits into the desired category. A simple solution to these difficulties is obtained by sorting jobs into streams as they enter the computer. For scheduling purposes there is need for a computer and a tape stream and for various peripheral streams. The tape and peripheral streams supply jobs which will maintain steady use of the tape decks and peripheral devices whilst computer stream jobs keep the central computing unit active.

There are several factors which influence the types of jobs which enter the various streams; these are now listed and their effects on the composition of the streams discussed.

1. The most obvious way of classifying jobs into streams is according to whether they are computer, tape or peripheral limited.
2. With short jobs where execution time is less than some specified time (say five minutes) there is a strong possibility that estimates for the use of peripherals, tapes and the central computing unit are inaccurate. This is particularly true of "common" jobs which contain no job description of their own but use the standard Atlas one.
3. If long and short jobs occur in the one stream, there is the possibility that a large number of short jobs may accumulate during the execution of a long job and this is undesirable.
4. Since a job's output may occur in uneven bursts, a long job, even if it is peripheral limited, cannot be relied upon to maintain steady use of the peripherals.
5. For reasons already mentioned peripheral stream jobs are to be executed in series and hence they must be short in order to ensure that all the peripherals are regularly supplied with output.
6. Since there are only a finite number of tape decks there is need for a tape allocation routine which governs the allocation of tape decks; this is most easily implemented if all jobs which require tape decks fall into only one or two streams.

Taking all these factors into account, a reasonable solution as to the composition of the stream lists, which fits in with the rest of the scheduling system, appears to be:

1. All short jobs which do not use tapes are allocated to the stream belonging to the peripheral which they use most.
2. All short jobs which use tapes and all long jobs where tape time exceeds their computing time are allocated to the tape stream.
3. Remaining jobs, that is all long jobs which are not tape limited, are allocated to the computer stream.

Jobs are sorted into their streams as they enter the computer according to the information given in the job description. When a particular type of job is required on the Execute List, the appropriate stream is consulted and the first job chosen; by this means the search time is reduced to a minimum and there is no possibility of any jobs being overlooked. When a computer or tape job leaves the Execute List it is replaced by a job from the computer or tape stream. When a peripheral job ends and there is a peripheral whose backlog is below a certain desired level or when there is no peripheral job on the Execute List and the output backlog on any peripheral falls below an emergency low level, then a job from the relevant peripheral stream is entered on the Execute List. Also, every time a new job joins a stream

queue and becomes available for execution the scheduler is consulted to see if this job is wanted on the Execute List. Once the tapes for a short tape job have been mounted, and it is not entered in the Execute List as a tape job, then it may be treated as a peripheral job and selected for execution on this basis; this affords a means whereby short tape jobs may by-pass long tape jobs. Some peripheral stream jobs may be willing to have their results output on any one of a number of peripherals and jobs in this category are queued in an "any" stream. In this case the scheduler outputs the results on the peripheral which is being used the least, thus helping to spread the output load evenly over as many peripherals as possible.

Since the stream queues must contain entries for every complete job in the computer they may become very long. It is essential therefore that entries in these lists be as short as possible to permit efficient scanning by the scheduler and tape and space allocation routines without excessive use of central computer time, core store and transfers to and from the drum. The simple method of choosing jobs in order from the stream queues for the Execute List makes it possible to implement the system if each entry only contains a condensed form of the job title, some reference position of the job on tape, and a link with the next job in the stream. However, jobs have to pass through an assembly stage (see next section) before entry to the Execute List and it is convenient if there is sufficient information (i.e. number of tape decks and system tapes required) in the stream queue to tell whether it is possible to go ahead and prepare the job. Also, in certain special circumstances it may be necessary to execute jobs out of their natural stream order. For these reasons approximate estimates of execution time, amount and type of output and space requirements, the type of compiler to be used and any external priority of the jobs are kept. This information is also valuable to the routines which govern the allocation of tape decks and space. The whole of this information is kept in 120 bits, which means that a single block of store (512 words) is sufficient to list over two hundred jobs which may be present in the computer at any one time. Precise information (names of tapes and system documents, location in store and on tape of each document, etc.) necessary for the assembly and preparation of jobs is contained in an "internal" job description, which is formed at input time from the programmer's job description and is recorded in an input stream as a separate document belonging to this job.

The question arises as to the action taken when a peripheral or tape becomes free and there are no jobs awaiting execution in that particular stream. Under the present stream structure it is possible that there are jobs in other streams which may make a great deal of use of this peripheral or tape deck; this is the case, for instance, with long peripheral limited jobs which have been placed in the computer stream. When a stream becomes empty it therefore appears worthwhile to search and see if there are jobs in other streams which could profitably be transferred to the empty stream. However, the fact that the stream is empty means that this peripheral is not likely to be in great demand in the near future and hence there is no point in transferring jobs, which make little use of the free peripheral, to the empty stream; in fact doing this could lead to inefficient operation. In order to ensure enough work for the central processor there should always be at least two jobs on the Execute List, even if it means taking jobs from the same stream.

ASSEMBLY OF JOBS

The computing speed of Atlas is such that small problems may be expected to occupy the central computer for only a few seconds during their execution. It is therefore essential that an entry in the Execute List should be able to be replaced rapidly by the next problem selected by the scheduler, and one of the tasks of the scheduler is to assemble problems in advance so that compiling and execution may begin immediately and are not subject to delays. The scheduler in fact selects in advance problems which are to be assembled, deals with any long-term preparations and then transfers these from the list of available jobs to an Active List, which comprises jobs in the course of assembly prior to execution. Normally computer and tape jobs are not entered on the Active List till the previous computer and tape jobs are finished; since these jobs are long it is difficult to predict when the previous job is likely to finish and a

short delay between the running of these jobs can be tolerated. Peripheral jobs are entered on the Active List when they are likely to be selected for execution, and where possible a certain minimum of jobs are kept on the list to ensure a ready supply of prepared jobs for the Execute List. When a problem is required for execution, only those problems on the Active List whose assembly is complete are considered, with the exception of when the central computer can only be used to start execution of a partially assembled problem.

Assembly of a problem involves collecting all the information required to run a problem so that it is available to the central computer with the minimum possible delay when the problem is entered to the Execute List. A completely assembled problem on the Active List has the relevant input information in the input well in the combined core and drum store, the required compiler or input routine in the drum store, and any private magnetic tapes mounted and their titles checked. The process of assembly may therefore consist of reading input information from the system input tape, reading documents previously recorded on previous system tapes ("document" tapes), reading a compiler into store where necessary, instructing the operator to mount magnetic tapes, and verifying the titles of all such tapes. The system may be easily extended to permit "on-line" use of other peripheral equipments which would be treated as being in the same class as magnetic tapes and prepared for use in a similar manner.

It is essential to observe that assembly of a problem may extend over a long period of time, although, of course, the central computer is only occupied for a small proportion of the time. Collection of a document from the system input tape may not be completed until a full "swing" of this tape has elapsed, a time interval, in the worst case, of around 16 seconds. Since the same tape is used to both write blocks received from input peripheral devices and read back previously written blocks to recover documents as they are required, the tape performs a "swing" action in which frequent scans are made over a few feet of tape, although it will gradually progress forwards. Collection of documents from other system tapes may require scanning over long sections of tape, and a delay of up to 5 minutes may be experienced even when the correct tape is already mounted. The mounting of magnetic tapes by the operator may be expected to occupy a variable length of time, but one which is necessarily long considering the speed of the central computer. The logic of the scheduler and the assembly routine takes into account these wide variations in assembly times, and attempts to minimize the effect of delay in any branch of the system on the other branches of the system which can operate concurrently.

The process of assembly is divided into two main phases. The first phase deals with long-term assembly of magnetic tapes, documents from system tapes other than the system input tape, and compilers. Not until this phase is completed is a job made available to the scheduler for inclusion on the Active List, and not until called for by the scheduler is the assembly of any documents from the system input tape initiated. The space and tape allocation routines, to be described later, scan the complete job list and initiate the first assembly phase for jobs near the head of the stream queue. As tape mechanisms become available, the operator is instructed to mount private tapes and/or system tapes; the required documents are read from system tapes into main store and from there may be temporarily recorded on the system dump tape. The space allocation routine makes main store available for compilers, which, if they are not already in main store, are read as required from a system library tape. As this stage of assembly of each problem is completed, the scheduler is activated, and should the problem be required on the Active List, any relevant documents are collected from the system input tape. The problem is, in fact, entered to the Active List, and is scanned by the routine controlling the system input tape; this routine collects documents required by any job on the Active List as the system input tape moves forward to the writing position, and hence can assemble several problems during one "swing" of the system input tape. The location of documents is obtained from the internal job description which, if necessary, is read in from the system input tape on the backward swing. It should be noted that any or all of the phases of assembly are omitted where they are not required. Frequently a problem will require only information recently input

through the input peripherals and will require a compiler already present in main store, in which case the first assembly phase is omitted. If demands on store are not heavy, the problem may also be already in main store, in addition to being recorded on the system input tape, and in this case, assembly is virtually instantaneous.

INTERACTION WITH OPERATORS

The scheduler can be independent of any operator scheduling but there does exist a two-way communication system between the operator and scheduler. When the scheduler requires action by the operator, such as the mounting of magnetic tapes, an explicit command is printed out to the operator. Because of the comparative slowness of operator action, requests to the operator are given, where possible, well in advance of when the action is needed. In fact, it is advisable that jobs which require some operator action should be fully prepared before entering the execution phase, otherwise inefficiencies resulting from a prolonged delay in operator action can be serious.

It is possible for the operator to convey information to the scheduler and an example of this is the allocation of an external job priority. From the scheduler's viewpoint external priorities given to a job through the operator at the user's request are necessary to satisfy the user, but may lead to gross inefficiencies in computer operation. The need to obtain the operator's approval before allocating a job special priority affords some protection, but it should be recognized that external priorities impinge on the scheduling system and determine independently what is to happen. Of course, the change of state in the computer brought about by the allocation of an external priority is taken into account by the scheduler when determining the best course of action in the future. Also, the framework of routines set up to implement external priorities may be used by the scheduler to achieve its own ends.

The normal scheduling system on Atlas is based almost solely on efficiency considerations and allocation of an external priority to a job by the operator affords the means whereby the individual user's requirements may be satisfied. There appears to be a need to allow four priorities, top, high, normal and low, with the following functions:

1. Top priority: the specified job is executed and the results output as soon as possible, regardless of the state of the computer, or what other jobs are awaiting execution.
2. High priority: the specified job jumps to the head of its stream queue.
3. Normal priority: the specified job is given the priority and treatment described in previous sections.
4. Low priority: the specified job is treated as normal until it reaches the execution list when it remains at the bottom of the list.

TAPE ALLOCATION ROUTINE

The Atlas operating system normally requires three tape decks for its own use, viz: input tape, output tape and dump tape; this means that three decks are occupied permanently by the supervisor, though the system is flexible enough to operate with fewer decks when absolutely necessary. When only two decks are available the input and output tapes are combined on one tape. Both tapes are normally used in a similar fashion, and the cost of combining them into one is a reduction in the effective size of input and output wells on tape. If only one deck is available this must be used by the dump tape which acts as an extension of main store. The operating system still functions when no decks are available, but only as efficiently as circumstances will allow, and care must be taken that input and output wells are not allowed to occupy all the free space in main store. If a computer installation has a large number of decks then it may be worthwhile for the operating system to use more than three decks; for instance, it would be valuable to have one deck permanently loaded with the library tape and another with a separate dump tape to record error monitor dumps of object programs.

Part of the information which accompanies each job into the computer is a list of the tapes required and any documents which have to be obtained from system tapes. Thus, before a job commences execution the number of decks it requires and the names of the tapes to be mounted are known; this information is valuable for scheduling purposes and the assembly of jobs prior to execution. However, in certain cases tapes are only required during part of

their total time of execution and thus tape decks can be called for or made free during the course of a job's execution; one common example is the use of tapes as temporary working space during compilation.

It is the job of the tape allocation routine to allot tapes to the various sources which require them in such a way that the scheduler can maintain efficient operation of the computer. There is clearly strong inter-action between the scheduler and tape allocation routine, the behaviour of one affecting the decisions of the other. The normal order of priority for allocating tape decks is as follows:

1. Supervisor system tapes.
2. Jobs or compilers which call for tapes during execution.
3. Jobs called to enter the execution list and requiring either private tapes or documents from system tapes.
4. Jobs near the head of their stream queues which require private tapes or documents from system tapes.

Since jobs have not direct access to system tapes and it is the supervisor's job to collect documents from these tapes, they can be dismounted as soon as the relevant documents are read into main store. However, there is no point in dismounting a system tape if the deck is not required, and if the deck is required it is worthwhile before dismounting the tape to read off any other documents which are likely to be called for in the near future. Exactly how many documents should be read into main store depends on the size of the documents, the state of the dump tape and the present demand on tape decks. If the space occupied by documents which are read into main store before they are called for is wanted, then the documents are recorded on the dump tape. It should be noted that apart from this reason the dump tape is used to record error monitor dumps, suspended programs and overflow from main store, which could include such items as jobs in process of execution and documents or output which lie outside the scannable region of the output tape.

SPACE ALLOCATION ROUTINE

Every block of main store space in Atlas is a member of a group, depending on its owner, and there are six main groups of owners; these are:

a) Free space
b) Input and Output wells
c) Jobs that are being prepared for execution
d) Jobs being executed
e) Compilers
f) The supervisor

The basis of the space allocation routine is that a hierarchy is formed of all these owners and also of all requests for space, and no request is allowed to take space from an owner of higher priority. The ratings of each member of this hierarchy are essentially dynamic, depending on the state of the machine at the time.

There is an area of space in the machine which is readily available to any request, this consists of all free space and all blocks in input and output wells which have been duplicated on magnetic tape, copies of which have been kept in main store. The number of these available blocks is known at any time. Essentially, a request can only take space from this group, and if there is not sufficient space, then routines are activated which begin to free blocks of a lower priority than the request. When this happens, the request is put into the space request queue, and each time a block is freed, the top element in the queue is checked to see if there are enough blocks for it. In certain cases the request is not queued but the space allocation routine returns to the routine which made the request with the information that the request has been refused, thus allowing the routine to take alternative action. For instance, the scheduler may ask for space to enter a peripheral job in the Execute List; if insufficient space is available, it may be possible for another smaller job on the Active List to fit into the space available. Certain other requests are allowed to by-pass this main queue, these are the types of requests which asked for single blocks at relatively infrequent intervals, and, if the request is not granted, some part of the computing system would be halted, e.g. if a request for input well space is not granted, then a peripheral may be halted.

The ordering of the space request queue is important because low priority requests for a large amount of space should not be allowed to block small, relatively high priority requests. Of course, care must be taken that a request is not always by-passed, and if it has been in

the queue for more than a certain length of time it is allowed to begin to reserve space. Also, the priority of most requests will be proportional to the length of time that they have been in the queue.

There are two types of information in main store, the first is information which has no copy on magnetic tape, i.e. programs being executed. In order to free this type of space it is necessary to dump the information on the dump tape. This can take a long time and owners in this class have a fairly high rating. However, the second class is information which is duplicated on magnetic tape, i.e. compilers; this information can be lost immediately.

It is a complicated task to keep track of the information in the store and on the dump tape, but a directory system has been devised which makes it possible to overwrite peripheral stream information in units of one block. However, for other information it is necessary to dump or overwrite it in reasonably sized sections, i.e. even if only a few blocks are needed an entire document might be overwritten.

The actual routines which do this work are in two sections. One section is in the fixed store, and this is sufficient to take care of most requests for space. The other section is in the main store and this contains the longer routines which deal with dumping and retrieving information. These space allocation routines, and in fact the whole supervisor, have been written in such a manner that it is possible to overwrite the major part of the main store routine, leaving virtually the entire store available to the user.

CONCLUSION

It must be emphasised that the system outlined above is suitable for any type of Atlas installation and is independent of the configuration of peripherals, core store and drum store, apart, of course, from changes in essential parameters at different installations. However, the scheduling system has been designed in two parts. Routines called into action most frequently are held in the fixed store, and these will be the same on all installations. They are called into action and effectively controlled by routines in the core and drum store and by parameters in the subsidiary working store, which can be changed in the light of experience, and to meet any particular requirements.

REFERENCES

1. KILBURN, T., HOWARTH, D. J., PAYNE, R. B. and SUMNER, F. H. (1961) "The Manchester University Atlas Operating System, Part I: Internal Organisation," The Computer Journal, Vol. 4, p. 222.
2. HOWARTH, D. J., PAYNE, R. B. and SUMNER, F. H. (1961) "The Manchester University Atlas Operating System, Part II: User's Description," The Computer Journal, Vol. 4, p. 226.
3. KILBURN, T., PAYNE, R. B. and HOWARTH, D. J. "The Atlas Supervisor," Proc. E.J.C.C., Dec. 1961.

DYSAC: A DIGITALLY SIMULATED ANALOG COMPUTER

J. R. Hurley
Senior Engineering Analyst
Allis-Chalmers Mfg. Co.
Milwaukee, Wisconsin

and

J. J. Skiles
Professor of Elec. Engr.
University of Wisconsin
Madison, Wisconsin

SUMMARY

A digital computer program which simulates a large electronic analog computer has been written for the CDC 1604 digital computer. In addition to providing many non-linear computing elements rarely found in electronic analog computers, the program accepts the input data in a form which may be written down directly from a block diagram or analog computer wiring diagram. Graphical output in the form of plotted curves is available by use of a digital plotter. The simplicity of the input language permits the program to be used easily by persons having no digital computer experience. This digital computer program, called DYSAC, an acronym for *D*igitall*Y* *S*imulated *A*nalog *C*omputer, is, in reality, a complete programming system, and as with FORTRAN, has a special language to facilitate its use.

Introduction

A large class of engineering problems involving the dynamics of systems is far more easily attacked by simulating the physical system on an analog computer, than by numerical integration of a set of differential equations. The factor which favors the analog machine is that it consists of separate components each performing some particular mathematical function continuously in time. This continuity of operation permits basic analog components to be arranged in groups which simulate complex systems by individually simulating portions of the physical system. Thus a minimum of effort is required for set-up and a physical significance may be attached to the signals at various points in the simulation.

The convenience of analog methods has prompted the development of digital computer programs which simulate the operation of analog computers yet retain the accuracy of digital devices. Selfridge[1] has done some of the basic work in this area. His original approach has been extended by Lesh and Curl into the DEPI[2] program. The existing simulator programs vary considerably in the degree which the input language is related to an analog computer. For example, considerable additional digital coding must be done by the user of DIDAS[3] to set up a problem. On the other hand, Stein, Rose, and Parker[4] have devised a digital compiler for the IBM 704 which employs FORTRAN as an intermediate in the compilation. It accepts an input language tailored expressly for the Electronics Associates PACE analog computer.

The DYSAC program to be described here, was written by the first-named author for the Control Data Corporation 1604 computer,* and was designed with particular emphasis on the simplicity and clarity of the input language.

Description of the DYSAC Program

The DYSAC program has been written so that, from the user's viewpoint, there are supplies of analog computer components available

* The University of Wisconsin's CDC 1604 is a high speed, 32K core machine with four magnetic tape units. Typical speeds are 7.2 microseconds for add, 36 microseconds for floating multiply. The 1604 uses a 48 bit wordlength with 2 instructions per word.

which may be interconnected according to directions supplied by the user. The 1154 components available include integrators, summers, limiters, square root generators, sine and cosine generators, natural logarithm and exponential generators, arbitrary (tabular) function generators of single variables, transport delay units, dividers, and relays as listed in Table I. Multiplication is performed as an auxiliary operation by the integrators and adders.

The input data to describe a problem to the DYSAC program includes the specification of the inputs to each DYSAC analog computing component and is called the patching. (This corresponds to physically patching the components of an analog computer.)

The "patching" information, the numerical data for the problem, and certain auxiliary data are used by the DYSAC program to obtain a numerical solution to the problem.

The numerical methods that are used in the DYSAC program for digital simulation differ from those inherent in analog computation with conventional analog computers in several important ways. In conventional analog computers the outputs of all computing components are calculated simultaneously and continuously in time. In a DYSAC simulation, the outputs of all analog components are calculated for discrete increments in the value of the independent variable (usually time.) For each increment in the independent variable, new outputs for each DYSAC component are calculated *in sequence* in the order specified in Table I; after new outputs for all components have been calculated (this constitutes an iteration), the independent variable is incremented and the process repeated iteratively until the program reaches a predetermined stopping point.

The DYSAC Input Language

The input data required to describe a simulation have been grouped into seven sections, as given in Table II.

Of the seven data sections required for a problem, five are straight-forward and their purpose fairly obvious. All sections of the data are presented on standard IBM cards. The title and headings are merely alphanumeric data used to identify problems and output values. The potentiometer settings, initial values on integrators, and arbitrary tabular functions comprise the numerical data.

Section 7, supplementary machine language instructions, is not generally used, but provides a method of introducing a sequence of computer operations at the basic machine language level. This sequence is executed once each time the main program does one iteration.

TABLE I

DYSAC COMPONENTS

Components	*Serial Numbers*	*Quantity*
Integrators	N01 to N99	99
Adders	A01 to A99	99
Limiters	L01 to L50	50
Sq. Root Generators	Q01 to Q50	50
Sine Generators	S01 to S50	50
Cosine Generators	C01 to C50	50
Log. Generators	G01 to G50	50
Exp. Generators	X01 to X50	50
Function Generators	F01 to F50	50
Time Delay Units	T01 to T12	12
Dividers	D01 to D99	99
Relays	R01 to R99	99
Potentiometers	P01 to P99	99
"	H01 to H99	99
"	J01 to J99	99
"	K01 to K99	99

TABLE II:
DYSAC INPUT
DATA SECTIONS

1. Problem title.
2. Patching. Description of connections between components.
3. Initial values for integrators.
4. Potentiometer settings.
5. Function tables.
6. Headings.
7. Supplementary machine language instructions.

TABLE III:
CONTROL OPTIONS

TITLE
RETAIN TITLE
PATCHING
RETAIN PATCHING
INITIAL VALUES
CLEAR INITIAL VALUES
RETAIN INTEGRATOR VALUES
POT SETTINGS
RETAIN POT SETTINGS
FUNCTION TABLES
RETAIN FUNCTION TABLES
HEADINGS
RETAIN HEADINGS
MACHINE LANGUAGE INSTRUCTIONS
NO MACHINE LANGUAGE
RETAIN MACHINE LANGUAGE

Section 2, patching, describes the interconnections between the hypothetical analog computing elements available in DYSAC. This description is formulated by giving the inputs to each component used. The components are described in the order given in Table I. The serial numbers shown in this table are used in the descriptions. The method can be best explained by illustration. The "patching" for integrator N01 could be:

$$N01 = A01A02 + A03.$$

This is read, "the input to integrator N01 is (the output of adder A01 times the output of adder A02) plus (the output of adder A03)". Note that multiplication is performed at the input to the integrators. The patching structure for adders is identical. The equal and plus signs used in the patching have no operational significance, they serve merely as punctuation. If the input of N01 was to be the product of the outputs of A01 and A02 minus the output of A03, the standard plus sign must still be used in the patching and thus the patching would be written:

$$N01 = A01A02 + A03P05.$$

where P05 is a potentiometer set to a value of -1. Another important point is illustrated in this patching example. Pots are not considered as computational units with inputs and outputs, but are merely constants and can be used as reference values or in conjunction with multiplication as coefficients for variables. There is no limit on the number of terms which may be summed at the input of an adder or an integrator, but the number of factors in each term of the sum is restricted to 20 or less. DYSAC

adders and integrators do not cause an algebraic sign reversal as do their usual electronic analog counterparts.

The patching for other components is fixed in form. For example, a time delay unit accepts only two input signals, one providing the signal to be delayed and the other the amount of the delay:

T01 = N02A21.

Here the output of time delay generator T01 is the output of integrator N02 delayed by the amount given by the output of adder A21.

Operation of the DYSAC Program

The major part of the DYSAC program involves dissecting the alphanumeric input data and converting it into the proper machine language instructions. Since operations of this type are not generally feasible with the FORTRAN compiler available for the CDC 1604, the DYSAC program was written in symbolic machine language. In its present state, DYSAC operates as an interpretive program, that is, during each iteration in the numerical solution the interconnections between simulated analog computing elements are scanned by the program to determine the proper calculation sequencing. After the data describing the analog configuration of a problem is read by DYSAC, one initializing pass is made upon the patching information in order to reduce it to a sequence of memory location addresses. This sequence can be very efficiently and rapidly "interpreted" repetitively by DYSAC, thus the low net computing speed usually associated with an interpretive routine is greatly increased without sacrificing the inherent flexibility of the interpreter.

All the DYSAC components of a given type are "serviced" in order, starting with the adders and progressing to the relays according to the order in Table I. Machine language instructions, if present, are processed next. The integrators are then processed and the sequence continues again with the adders, etc. The program is divided up into sub-sections which each process a particular type of component. During the development of the numerical results of a simulation problem, the mathematical operations implied by the patching description for each component must be performed at each iteration. In order to save computing time during this interpretive iterative solution, the entire patching sequence is translated once by the DYSAC program into a more easily used form before the numerical solution starts. The sequence of serial numbers and punctuation marks is translated into a sequence of numbers by replacing each alphanumeric serial number with the address of the memory location in which the output of that component is stored. Certain numerical codes which can not be confused with permissible component output addresses are used to replace the plus signs and periods which occur in the patching. Serial numbers appearing to the left of an equal sign as well as the equal signs themselves, are deleted from the translated patching sequence. However, a count is made of the total number of each type of component defined in the patching. Then, since all the components were described in order, there can be no ambiguity concerning which component is described by any portion of the sequence of translated patching.

The translated patching sequence is used by the interpreter at each iteration during the numerical solution. The operation of the program can be shown, in principle, by considering the processing of the hypothetical dividers. The dividers follow the time delay units in the processing sequence. Patching for dividers is fixed in format. For example:

D01 = N01A02.

Here the output of divider D01 is to be the output of integrator N01 divided by the output from adder A02. After patching translation, numbers equal to memory addresses of the quantities which are to be the numerator and denominator occupy consecutive locations in the translated patching sequence. These addresses are taken from a patching table and inserted into "load accumulator" and "floating divide" instructions when the divider D01 is to be processed. These two instructions are then executed and are followed by an instruction which stores the result in the memory location allocated to D01. The program then counts having processed a divider, checks to determine if there are any more dividers in the simulation, and on this basis either proceeds to the portion of DYSAC which processes the next type of component (relays) or continues processing dividers. This is done by increasing the address of the "store output of D01" instruction so the next time it

functions, it will store the result in the location corresponding to D02. Then the index which is used as a tally to count through the translated patching is increased by 3 (translating the patching for a divider results in 3 consecutive words, a numerator address, denominator address, and a period symbol) and the process is repeated by inserting addresses taken from the patching sequence into "load accumulator" and "floating divide" instructions.

The processing of other components makes similar use of instruction skeletons as outlined for the divider processing.

The sine, cosine, logarithm, exponential, and square root functions are computed with the usual rational approximation subroutines and provide approximately 10 place accuracy.

Special Features

The time delay units produce true transport delays. The delay time can be a continuously variable quantity generated by some other component in the simulation. (This method of delay realization is far superior to DYSAC simulation of usual analog computer techniques such as the use of a Padé approximation, although such delay simulations are straight forward with DYSAC, if desired.)

The arbitrary function generators operate with tables which contain pairs of coordinates of points of the function. An extra degree of flexibility is provided by a feature which enables the function generators to operate with any of three methods of interpolation. The methods available are "1 point" or "boxcar", 2 point or linear, and 3 point or parabolic. Figure 1 illustrates these interpolation methods.

The relays operate when the output of a specified component exceeds that of a second specified component. When the relay operates, it will set the output of a particular pot or integrator equal to the current output of still another component. An example of relay patching is:

R01 = A01A02P08P09.

Relay R01 will set pot P09 equal in value to pot P08 when the output of adder A01 exceeds that of A02. Electronic analog computer relays are usually used to alter the connections between analog computing components. Even though DYSAC relays operate considerably differently than their electronic analog computer counterparts, they may be made to perform the same function by setting a coefficient pot to zero to "open" a circuit or by changing a zero pot to some non-zero value to "close" a circuit.

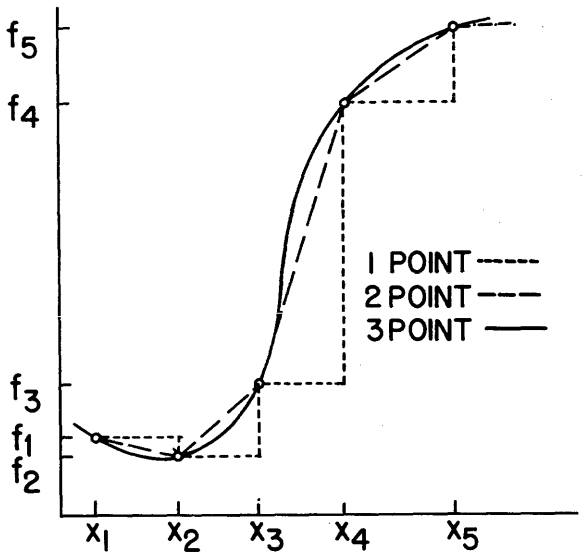


Figure 1. Function generator interpolation methods.

Four of the potentiometers (P01 to P04) have special functions during the solution of a problem. Pot P01 is automatically increased during the solution so that it always contains the current value of time (independent variable). P02 contains the value of the time increment used in the numerical integration algorithm. P03 contains the time value at which the solution is terminated. P04 determines the number of iterations performed between points at which the output of specified components are listed. All four of the special pots are subject to alteration by relays during the course of the solution.

When a series of problems is run, it often happens that only a few parameters are changed from the preceding problem. In order to save card punching effort it is possible to retain any of the main data sections (as given in Table II) from the previous problem. These possible variations in the operation of the program are regulated by 7 control cards, one preceding each of the 7 main data sections. Table III lists the various options available for the control cards.

The provision for inclusion of machine language level instructions as part of a problem

may seem to be strange since the purpose of DYSAC is presumably to avoid having to program in machine language. Actually this option was provided originally as a "last resort" method for implementing calculations and special function generation impossible with the normal DYSAC components; however, it has proven valuable for making experimental changes and modifications in the DYSAC program itself. This option is also quite convenient in the simulation of sampled-data systems where the discrete and digital functions may be realized directly by the proper machine language instructions, as is illustrated later in Example 3.

Magnitude and time scaling which are often bothersome necessities of electronic analog problem preparation are, for all practical purposes, not required when using the DYSAC simulator. Floating point arithmetic is used throughout, which provides a 36 bit fraction, a ten bit exponent, and a sign bit for each number (48 bit wordlength). Thus a number range from 10^{-308} to 10^{308} is covered and about 11 decimal places of significance is provided.

Generation of Special Functions

The DYSAC integrators and adders do not duplicate their analog counterparts in all respects. In particular, nothing in DYSAC corresponds to the summing junction of a conventional operational amplifier. Thus DYSAC cannot duplicate directly some of the standard analog computer techniques for realizing special transfer functions such as the appropriate connection of series or parallel combinations of capacitors, resistors, diodes, etc., to the summing junction. However, such transfer functions can usually be generated easily in other ways by a suitable combination of DYSAC components.

Some simulations may call for functions that are inconvenient, or impossible, to implement with available DYSAC components. Three alternatives are available:

1. Rewrite DYSAC to include the new function generator as an additional component everytime the need for a new function arises. This procedure would require the services of a skilled CDC 1604 programmer, familiar with the DYSAC program, to alter extensively the basic structure of the patchword conversion routines, and would necessitate a reassembly of the program after every modification. This procedure has not been used.

2. A simple machine language program using octal addresses and instructions can be written to realize the desired function. This program is then punched on IBM cards as supplemental machine language instructions and read into the 1604 with the patching and data cards for the problem. Differentiation, using the fundamental relation,

$$\frac{dx}{dt} = \frac{x(t) - x(t - \Delta t)}{\Delta t} = \frac{\Delta x}{\Delta t}$$

implemented in this manner requires the execution of 10 1604 instructions. A more accurate differentiation algorithm obviously can be implemented in this manner, if desired. (A differentiator can also be constructed to realize dx/dt by using a time delay unit to generate $x(t\text{-}\triangle t)$, a divider, and relays to set up initial conditions. However, the octal machine language differentiator program is executed considerably faster than the processing time for the differentiator constructed by patching the DYSAC components.) Octal machine language is the most commonly used and least costly method of generating special functions.

3. Use a previously written symbolic language program generating the required function. The original symbolic language DYSAC program is then reassembled with any new function programs as subroutines. From the new assembly, the octal addresses of these additional subroutines are noted for reference. If one of these special functions is required it is only necessary to use a few octal machine language instructions to enter and exit from the subroutine. This procedure does not require a time-consuming and costly modification of the DYSAC program logic; however, reassembly of the DYSAC symbolic program is necessary at a cost of approximately 3 minutes of 1604 time whenever new functions are added. These new functions then become permanent additions to DYSAC. (The additional circular functions of tangent, arctangent, arcsine and arccosine have been added to the original DYSAC program to facilitate coordinate conversions required in the solution of such problems as aircraft and missile dynamics.)

Accuracy Considerations

The operation of the simulator program is centered around the fourth order Runge-Kutta method of numerical integration. The choice of integration algorithm is necessarily a compromise between the desirable but conflicting requirements of accuracy, minimum computer running time, ease of starting a solution, and the ease with which the increment in independent variable may be changed during the solution. Although in comparison with Runge-Kutta methods, algorithms like the Adams-Moulton and other predictor techniques permit the use of larger increments in the independent variable $\triangle t$ as well as simplify error estimation during solution, a fourth-order Runge-Kutta method has been used in DYSAC primarily because of the ease both in starting solutions and in making changes in independent variable increment during solutions.

The value of $\triangle t$ must be sufficiently small that the Runge-Kutta process not only is stable, but gives acceptable accuracy. In general, the smaller the time increment, the greater the accuracy, but also the more computer time required. Cumulative round-off error due to very small increments is in general not a problem since the floating point number system used by the 1604 provides about 11 decimal places of significance. If the time increment is too large, the numerical integration method becomes unstable and solutions "blow up".

A certain amount of experimentation with the value of $\triangle t$ is often necessary when beginning unfamiliar simulations. As in selecting the time scale in conventional analog computer studies, prior knowledge of the approximate system natural frequencies and loop gains can be extremely useful in selecting a trial value of $\triangle t$.

The mathematical literature contains methods for approximating the error in the use of several integration algorithms, and suggestions for automatically changing the increment in independent variable to keep the error during each step within prescribed bounds. However, selecting a suitable error tolerance can be quite difficult, particularly for simulations involving many variables, since during portions of a solution relatively small errors may propagate to produce significant errors at later stages of the solution. Conversely, relatively large errors in some variables during certain portions of the solution may be inconsequential in their later effect on the solution. Different methods of numerical integration and possible ways for the DYSAC program to automatically adjust $\triangle t$ during solution to improve accuracy and minimize computer time are the subject of current study.

DYSAC Outputs

The output from a DYSAC digital simulation is: a) a printed tabular listing with seven output variables listed per page, 50 values per variable per page, with a maximum of 56 output variables, but with the number of output values for the variables limited only by number of iterations necessary to obtain the solution to the problem; b) graphical plots of the output variables cross plotted in any desired manner obtained by use of a digital plotter and auxiliary plotter programs.

Two output tapes are written by the 1604 during every DYSAC solution of one or a series of problems. A binary output tape contains only the raw numerical output quantities, with end-of-file marks separating data of different problems. The second, or printer tape, has the numerical results in BCD format with descriptive column headings and additional alphanumeric information as the title assigned to the problem, a complete printout of the patching, a listing of the settings of all pots and function table values. This printer tape is then printed off-line through a small CDC 160 computer using an Anelex 1000 line/minute printer.

The binary output tape is the input to another program, DYPLOT, which rearranges the numerical output, and generates the necesary commands to prepare curves on a CALCOMP Model 570 plotting system. The output of DYPLOT is a third tape, the plotter tape, which is read by the CALCOMP 570 tape unit and the curves then plotted on the CALCOMP 560R incremental digital plotter. The DYPLOT program requires certain additional information, such as the title of the graph, axis designations, scaling, etc., be supplied on punched cards by the user.

Illustrative example:

A block diagram of a simple control system is shown in Figure 2. Its purpose is to energize an inductive load with currents up to 10,000 am-

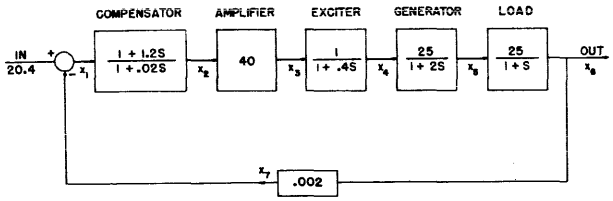


Figure 2. Block diagram of illustrative problem.

peres in response to input signals in the range up to 20 volts. The characteristics of the elements are given below.

Component	*Time Constant*	*Gain*
amplifier	0	40 volts/volt
exciter	.4 sec.	1 volt/volt
generator	2 sec.	25 volts/volt
load coil	1 sec.	25 amps/volt
sensing element	0	.002 volt/amp

The DYSAC solution is to show the response of the system to a 20.4 volt step input over a 7 second interval. The time increment selected for the numerical integration is 0.005 second and printout is desired every .02 seconds (every 4th iteration). Figure 3 shows one of the possible DYSAC configurations which will simulate the system of Figure 2. The complete DYSAC input data for the sample problem is reproduced from IBM cards in Figure 4. Figures 5 and 6 illustrate the two modes of output possible with DYSAC, the printed listing and the graphical output provided by the CALCOMP digital plotter.

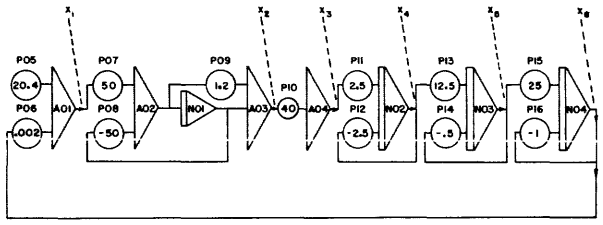


Figure 3. Dysac diagram of illustrative problem.

```
BCD 8 TITLE
BCD   CURRENT REGULATION SYSTEM
BCD 8 20.4 VOLT STEP INPUT
BCD 8 PATCHING
BCD   N01=A02.   N02=A04P11+N02P12.   N03=N02P13+ N03P14.
BCD   N04=P15N03+P16N04.
BCD   A01=P05+P06N04.   A02=P07A01+P08N01.   A03=P09A02+N01.
BCD   A04=P10A03.   OUT=P01.   OUT=A01.   OUT=A03.
BCD 8 OUT=A04.   OUT= N02.   OUT=N03.   OUT=N04.
BCD 8 CLEAR INITIAL VALUES
BCD 8 POT SETTINGS
DEC   0,,.005,7,4
DEC   20.4,-.002,50,-50
DEC   1.20,40,2.5,-2.5
DEC 8 12.5,-.5,25,-1
BCD 8 RETAIN FUNCTION TABLES
BCD 8 HEADINGS
BCD       TIME, SEC      ERROR VOLTS    COMP. OUT., V    AMP. VOLTS
BCD 8     EXCIT. VOLTS   GEN. VOLTS       LOAD AMPS
BCD 8 NO MACHINE LANGUAGE INSTRUCTIONS
END
```

Figure 4. Dysac encoding for illustrative problem.

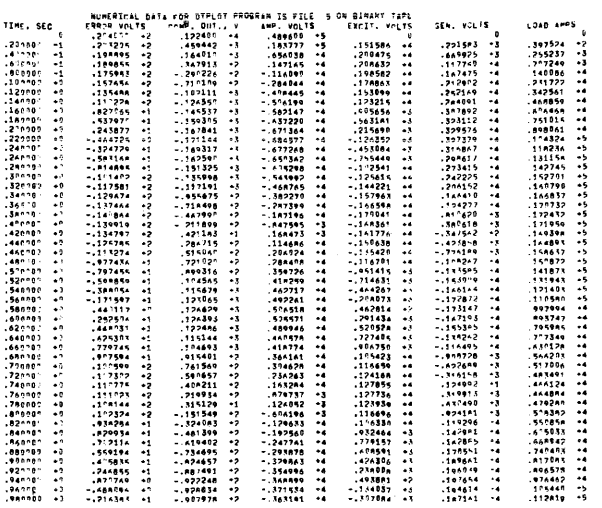


Figure 5. Sample output for illustrative problem.

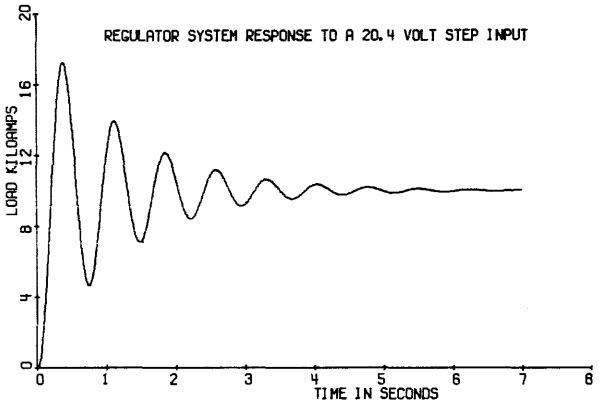


Figure 6. Sample plotter output for illustrative problem.

This simulation involving 1400 iterations on 4 adders and 4 integrators required 40 seconds of CDC 1604 computer time.

Examples of Simulations

Three broad classes of system simulations that are representative of the applications of DYSAC are:

1. Aircraft and missile problems.
2. Electromechanical transients in electric machines.
3. Chemical processes.

Example 1: An Aircraft Simulation

Aircraft and missile simulations are often characterized by the large number of analog computer components necessary to represent the airframe, autopilot, power plant characteristics, aerodynamics, coupling between control axes, and coordinate transformations required by the analysis. The drift and noise free character of digital simulations and the freedom from the usual scaling problems are distinctly of value in these simulations, which are often of linear systems or of systems with only one or two small nonlinearities. Large nonlinearities, when they do occur, can also be handled easily using DYSAC.

In a recent series of studies Grzelak[5] has made extensive use of the DYSAC program in studies of automatic landing systems for aircraft. In these studies the aircraft was presumed to fly a straight line extension of the glide slope of an existing ILS (Instrument Landing System) down to a point at or near the approach end of the landing runway, but at a height of 40 ft. above the runway. A flare computer continuously generated commands to the autopilot which caused the aircraft to follow closely a predetermined flare path and to land about 1500 feet down the runway, with a rate-of-descent at touchdown of 2 feet/second or less.

Of primary interest in these studies was the dispersion of the point-of-touchdown and of the rate-of-descent at touchdown as influenced by the choice of flare path controller and wind gusts. Comparative studies were made of the system performance with many types of flare paths, including exponential, circular, segmented straight-line, and terminal control, and some combinations of these flare paths. The studies were conducted using a simplified pitch-axis representation for a F94C aircraft with an E-10 autopilot because the necessary aircraft and autopilot data and some check solutions from conventional analog computer studies were available.[6, 7]

The block diagram of the system studied is shown in Figure 7. The system is completely linear for some types of flare paths, such as exponential, when the airspeed is assumed to be constant. Air speed computation and some types of flare path controllers resulted in nonlinearities that required the use of dividers and

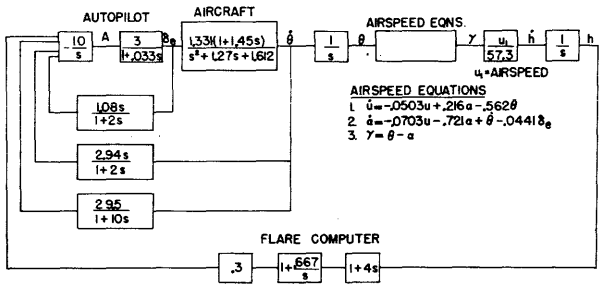


Figure 7. Block diagram of aircraft problem.

multipliers in the DYSAC representations of these situations. However, the problem was basically that of a linear system and the DYSAC representation for each combination of autopilot-aircraft and flare computer studied employed an average of 13 integrators and 15 adders.

Although the distance to the point of touchdown and the rate of descent at touchdown were of primary concern, additional information on the aircraft performance as a function of time during flare-out was also desired and the following quantities were included in the printout: time (from initiation of the flare), elevator deflection, pitch rate, pitch attitude, change in airspeed, angle of attack, glide angle, height rate (rate of descent), altitude, and distance (from beginning of flare to touchdown).

Approximately 8 seconds of flight time elapses from the beginning of the F94C aircraft flare to the point of touchdown, varying slightly with the type of flare controller and with the nature of the gust disturbances (which were introduced as step changes in the angle of attack or airspeed). The actual solution time on the 1604 computer for a complete flareout solution averaged about 10 seconds, using a value of time increment $\triangle t = 0.05$ seconds, with excellent accuracy in the results. This approximated real time operation. A time increment of 0.1 second could have been used with acceptable accuracy in the results, with even less computer time required per solution.

With such a short computer time required for a complete solution (approximately real time), many combinations of flare controller, airspeed and gust conditions were economically studied, at a computer cost of about $1 per case.

Example 2: Electric Machine Transients

The study of electromechanical transients in electrical machines is complicated by the wide

disparity in some of the electrical and mechanical time constants. In many studies it is convenient, and quite accurate, to assume the machine to be in a steady state condition either electrically or mechanically, thereby greatly simplifying the computer study. However, there are also many problems of interest where it is necessary to represent both electrical and mechanical transients. Digital simulation using DYSAC has been utilized by several University of Wisconsin staff members and graduate students in studies of induction and synchronous machines with non-sinusoidal applied voltages and with complex control circuits, including discontinuous controllers and silicon controlled-rectifiers. The digital simulations have been convenient, and in some cases, possible only because DYSAC handles nonlinearities so easily.

It is common practice to transform the transient electrical equations describing machines to reference frames convenient to the problem under study. Commonly used reference frames include the stator, the rotor, and a synchronously rotating reference frame.

With both electrical and mechanical transients considered, and neglecting saturation, the dynamic equations describing a two-phase, two pole, uniform air gap, wound rotor induction motor, referred to rotor reference axes α and β coinciding with the stator phase axes a and b, may be written as:

$$(R_{aa} + L_{aa}P)i_a + (MP)i_\alpha = V_a \quad (1)$$

$$(R_{aa} + L_{aa}P)i_b + (MP)i_\beta = V_b \quad (2)$$

$$(MP)i_a + (Mi_b)P\theta_m + (R_{xx} + L_{xx}P)i_\alpha + (L_{xx}i_\beta)P\theta_m = V_\alpha \quad (3)$$

$$-(Mi_a)P\theta_m + (MP)i_b - (L_{xx}i_\alpha)P\theta_m + (R_{xx} + L_{xx}p)i_\beta = V_\beta \quad (4)$$

$$\frac{Jd^2\theta_m}{dt^2} + T_L = T_f = M(i_b i_\alpha - i_a i_\beta) \quad (5)$$

where R_{aa}, L_{aa}, R_{xx}, L_{xx} and M are winding constants; i_α, i_β, i_a, and i_b the current variables; θ_m the angular position of the rotor; $P\theta_m$ the instantaneous speed; and T_L the load torque (a function of speed).

Flux linkage equations are probably more commonly used in machine analysis than equations (1) - (5), which are limited to those applications where saturation can be neglected (as is often reasonable). Both the flux-linkage form of the equations and equations (1) - (4) have been used in digital simulations.

Both equations (1) and (3) contain both Pi_a and Pi_α; similarly, equations (2) and (4) contain both Pi_b and Pi_β. Thus two of the same highest-order derivatives appear in each of two pairs of equations, a condition known to lead to "algebraic loops" (loops without integrators), and *instability* in analog computer setups. Because the variables are thus implicitly defined in terms of themselves, the Runge-Kutta integration method of DYSAC will usually also be unstable if equations (1) - (4) are programmed directly.

The undesirable implicit relations between the variables can be eliminated by eliminating either Pi_a or Pi_α between (1) and (3); and eliminating Pi_b or Pi_β between (2) and (4). The two resulting new equations are then programmed for solution together with a proper choice of two of the original equations (1) - (4), plus the torque equation (5). There are 16 different combinations of equations that can be derived in this manner.

Studies of electromechanical transients in both relatively high rotor resistance, small inertia servomotors and studies of multiple-horsepower, large inertia machines have shown that some of the methods of elimination give distinctly more accurate results for a specified $\triangle t$ than is obtainable with the other methods. We conclude that attention should be given to the method of elimination used in eliminating algebraic loops in any system of equations and that some experimentation with different methods may be worthwhile.

The nonlinear differential equations (1) - (5) give rise to variable loop gains with the appearance of speed dependent terms such as (Mi_b) $(P\theta_m)$, because both i_b and θ_m are variables that vary with time. Values of $\triangle t$ that result in stable Runge-Kutta numerical integrations for small values of speed ($\omega_m = P\theta_m$), can result in instability and obviously incorrect results at larger values of speed because of the larger loop gains.

In a series of studies of electromechanical transients during the starting of a 5-HP induction machine a value of $\triangle t = 0.001$ seconds gave excellent accuracy, while $\triangle t = 0.01$ seconds gave an unstable solution. It is characteristic of the Runge-Kutta process in unstable situations that the solution appears to be proceeding smoothly, and then diverges suddenly as the speed, and thus a loop gain, exceeds a critical value.

In a series of studies[8] on the starting transients in servomotors, with some of the nonlinearities in the machine represented, the simulations have used 8 integrators, 14 adders, 1 sine generator, 1 cosine generator, 4 function generators, 1 divider and 1 relay. The phenomena of interest lasted about 0.07 seconds and a $\triangle t$ of 0.0001 seconds was used to get acceptable accuracy. The DYSAC solutions for each case averaged about 50 seconds of CDC 1604 computer time.

Example 3: Sampled Data Systems

Chemical processes often have inherent transport time delays and DYSAC can be used readily for such simulations. Due to the ease with which nonlinearities may be handled, DYSAC is particularly valuable in conducting simulations of chemical reacting systems, where nonlinearities are nearly always associated with reaction kinetics. A simulation of an isothermal catalytic reactor which was approximated by 5 ideally mixed stages and involving only 3 chemical species required 20 multiplications and 5 divisions. Had the reactor temperature sensitivity been included, an additional 4 function generators would have been required. Turbulent fluid flow problems often require generation of nonintegral powers of flow rate in order to determine pressure drops. The logarithm and exponential generators of DYSAC handle these situations easily and accurately.

The inclusion of relays and transport time delay units in the DYSAC program (as well as the possibility of including discrete logical instructions using the machine language option) has resulted in a programming system well-adapted as a tool for the analysis and simulation of sampled-data or pulsed data systems in addition to continuous systems.

To indicate both the feasibility and ease of data preparation for such applications, a very simple sampled-data system and a digital controller was simulated via DYSAC. The process as shown in Figure 8 consists of a heater with a liquid flowing through it at a constant rate. On leaving the heater, the fluid flows through a long pipe exhibiting a 10 second transport delay. The long pipe empties into a perfectly mixed vessel having a 10 second retention time. The temperature is measured at the outlet of the vessel and is to be controlled by varying the heat rate to the heater. The controller is to be a sampled-data device which compares the desired and actual temperatures every 5 seconds, and makes an adjustment on the heating rate every 5 seconds. The controller is to operate so that the temperature error due to a step change in set-point is reduced to zero in a minimum number of sampling periods. The signal flow diagram for the sampled-data system is shown in Figure 9.

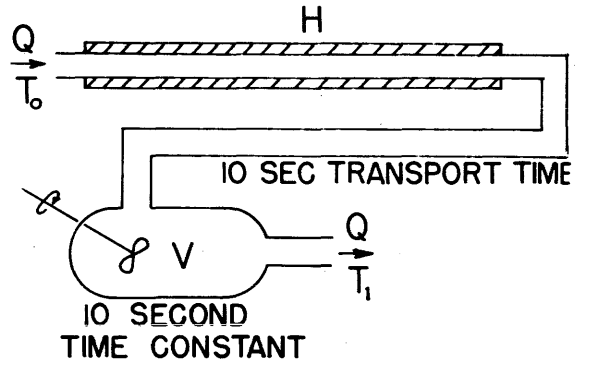


Figure 8. Sketch of transport process.

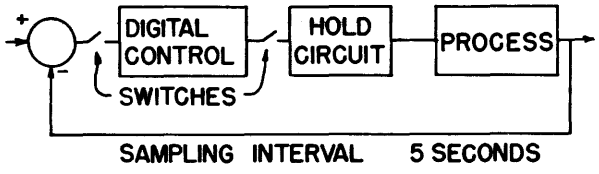


Figure 9. Signal flow diagram of sampled data system.

The Laplace transform of the process function is:

$$G_p(S) = .02 \frac{e^{-10s}}{1 + 10s} \quad (6)$$

Using the standard techniques of sampled-data system design,[9] the z-transform of a controller exhibiting the specified characteristics is found to be:

$$D(Z) = 127.07 \frac{(1 - .6065Z^{-1})}{(1 - Z^{-3})} \quad (7)$$

After the form of the desired controller was developed by analytic means, the entire system was simulated by DYSAC. The first step was the formulation of a basic analog computer diagram as shown in Figure 10. Four relays and an integrator can be arranged to create the sampling switch.

As mentioned previously, the possibility of simulating the controller function via machine language instructions exists. This is in fact an easy and efficient course of action to follow since in essence the digital control function is performed by the 1604 digital computer directly instead of being simulated by DYSAC. Changing equation (7) to a time domain recursion formula where c is the controller output and r its input:

$$c_n = c_{n-3} + 127.07r_n - 77.07r_{n-1} \quad (8)$$

To use this algorithm, the output values (c's) must be saved for 3 sampling periods and the input values (r's) for one sampling period.

A slightly different algorithm which closely parallels the analog flow diagram for realization of $D(Z)$ as shown in Figure 10 may be developed by arbitrarily defining a quantity e_n as follows:

$$e_n = \frac{1}{127.07}(c_n + 77.07e_{n-1}) \quad (9)$$

and thus

$$c_n = 127.07e_n - 77.07e_{n-1} \quad (10)$$

substituting for c_{n-3} in the right member of equation (8)

$$c_n = 127.07(e_{n-3} + r_n) - 77.07(e_{n-4} + r_{n-1}) \quad (11)$$

In comparing equations (10) and (11) it is obvious that

$$e_n = r_n + e_{n-3} \quad (12)$$

Thus a method of computing e_n from known quantities has been derived so that it fulfills the arbitrary definition of equation (9). Note that e_n corresponds to the output of the third adder in Figure 10. The method of realizing the digital controller involves saving only one quantity for 3 sample periods, a saving of 1 storage cell over the method of equation (8). Furthermore, only 19 CDC 1604 instructions are required to implement the controller and the sampling switch action, one less than required for the first method.

The response of this sampled data system to a 50 degree change in set point is illustrated in Figure 11.

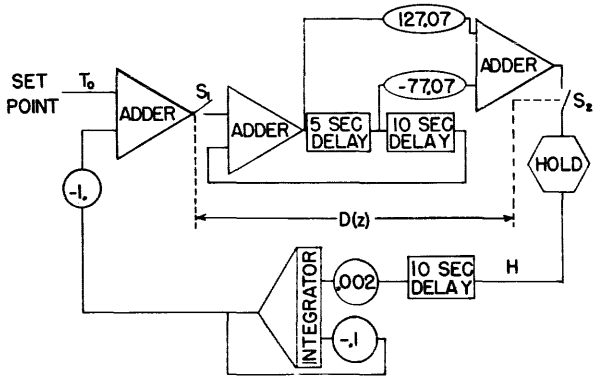


Figure 10. Generalized analog computer diagram.

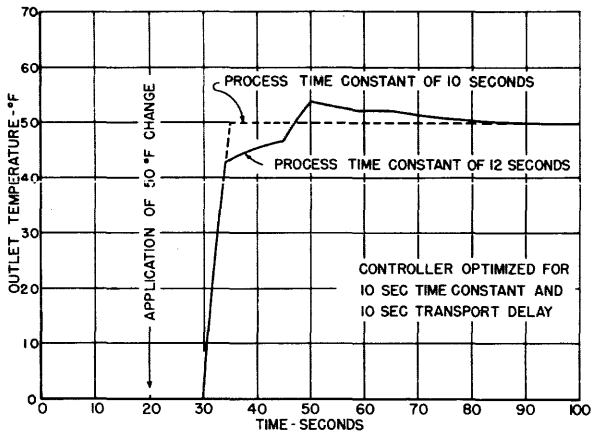


Figure 11. Response of transport process.

CRITIQUE OF DIGITAL SIMULATION

Advantages

1. The easy-to-learn coding structure of DYSAC enables individuals with no prior knowledge of either analog computing

techniques or digital computer programming to very quickly solve analog computer type problems.

2. It is not necessary for the analog computer type of connection diagram to be drawn for all problems. The DYSAC patching can be written down directly from the differential equation in many instances.
3. Retention of the block diagram representation of systems and components, while not necessary, is a desirable feature to many users. Conventional digital solution of system problems often results in some loss of identification of the system.
4. Elimination of the necessity for number scaling with DYSAC is a substantial advantage over conventional analog computer studies, particularly in the case of complex, nonlinear systems.
5. The problems of noise and drift in dc levels so familiar in analog computers do not exist with digital simulation.
6. Nonlinear operations such as squaring, cubing, forming sine, cosine, exponential, logarithm functions, and multiplication present no special problems in digital simulation with DYSAC.
7. The availability of true representation for time variable transport delays is a real advantage over most of the analog computer techniques that use truncated expansions of various functions to approximate, and often poorly, transport time delays.
8. The patching and data for a problem can be completely and inexpensively checked before going on the computer. While punching and checking program and data cards may take an appreciable time for a large simulation, the digital computer is not required.
9. Individual case studies can be run with practically no setup time once the program patching deck is available. (This compares favorably with the use of separate patch panels, tape set pots, etc., in large analog computer installations.)
10. Studies of many different problems can proceed almost simultaneously, unlike with an analog computer installation where one complex simulation may tie up the computer for several weeks.
11. The floating point arithmetic used in DYSAC can be used to achieve accuracy in problem results that is completely unattainable with an analog computer.
12. Graphical outputs from digital simulations are easily prepared, are accurate, and are readily reproducible when using a digital plotter such as a California Computer Products Model 570 off-line magnetic tape plotting system.
13. The ability to do large scale analog-computer type simulations on a digital computer is attractive to many users having good digital computing facilities and little or no analog computation capability.

Disadvantages

1. A DYSAC solution to a problem will require more *computing* time than would be required for a conventional digital computer program written by a skilled programmer to run on the same computing machine. (However, the time required to program DYSAC will be much less than the time necessary to write a conventional program to solve the same problem.)
2. Some flexibility is lacking in DYSAC, as in similar programs, because the user is usually limited to only those components available in the program. The provision in DYSAC for special machine language instructions substantially improves the flexibility of the program.
3. In its present form, neither DYSAC nor similar programs have provision for the introduction of real-time data from external hardware, as is done in some analog computer simulations.
4. The choice of the value of $\triangle t$ to be used in a simulation is a compromise; a small value of $\triangle t$ is desirable for accuracy, while a large value minimizes computer rental cost. While rule-of-thumb guides can be used to determine an initial value of $\triangle t$ to be used for a given problem, no procedure exists for manually or automatically selecting a value of $\triangle t$ to give a prescribed accuracy to all output quantities of interest at all times during a simulation. Some experimentation with the value of $\triangle t$ is often necessary when studying a new system of equations.

5. Because of the expense of computer time, digital simulation does not permit the type of experimentation with pot settings and immediate observation of the results that is possible with analog computers, particularly for repetitive type computers.
6. To avoid unnecessary computer rental, decision making and data changes while actually running problems must be discouraged.
7. The computer running time and thus the cost of digital simulations varies almost directly with the number of components in the simulation, in contrast with analog studies where computer running time is not a function of the number of components. (However, rental costs for an analog facility will increase with the size of the facility.)
8. As presently written, DYSAC is not readily adaptable to random process studies, although Hurley[10] has made a number of simulations of an adaptive process controller using a machine language random noise generator.

CONCLUSIONS

Digital Simulation should be considered as an extremely valuable supplement for an analog computer, but not as a complete replacement. It will prove very useful where inadequate analog computer facilities exist to solve a particular problem, whenever more accuracy is desired than can be obtained from an analog computer, or when it is necessary to obtain results in less elapsed time than would be required for either a conventional analog or a digital computer solution. Although Digital Simulation is usually excessively time consuming and costly on small digital computers, it is economically attractive on computers in the CDC 1604 and IBM 7090 class.

The DYSAC program is receiving wide usage at the University of Wisconsin and research is continuing both in the development of improved digital simulation techniques and in applications in many areas of engineering and science.

ACKNOWLEDGEMENTS

The DYSAC program was developed as one phase of a continuing project on digital simulation, under the direction of Professors J. J. Skiles and V. C. Rideout, Department of Electrical Engineering, University of Wisconsin, with National Science Foundation support under grant G19886.

Early phases of the "Digital Simulation" project were supported in part by the Research Committee of the Graduate School from funds supplied by the Wisconsin Alumni Research Foundation.

We wish to acknowledge the many helpful suggestions and comments from Professor V. C. Rideout, who is participating in the Digital Simulation studies.

REFERENCES

1. Coding a General-Purpose Digital Computer to Operate as a Differential Analyzer, R. G. Selfridge. Proc. of the 1955 Western Joint Computer Conference (IRE).
2. DEPI: An Interpretive Digital Computer Routine Simulating Differential Analyzer Operations, F. H. Lesh and F. G. Curl. Jet Propulsion Laboratory, California Inst. of Tech., Pasadena, Cal., March 22, 1957.
3. DIDAS, A Digital Differential Analyzer Simulator, G. R. Slayton. Twelfth National Meeting of the Assoc. for Computing Machinery, June 1958.
4. A Compiler With An Analog-Oriented Input Language, M. L. Stein, J. Rose and D. B. Parker, Proc. of the 1959 Western Joint Computer Conference. (IRE).
5. A Comparative Study of Flare Paths for Aircraft Automatic Landing Systems, T. Grzelak, MS Thesis, University of Wisconsin, June 1962.
6. Automatic Flare-out for Landing, D. Markusen, R. McLane and O. Pomeroy, WADC Technical Report 55-506, March 1956.
7. Flare Out Program for Air Force Type E-10 Automatic Pilot, R. V. Gaertner, R. C. McLane and V. Baxter, WADC Technical Report AD 5501-TRI, August 1957.
8. Transient Response Characteristics of an AC Servomotor, Ph.D. Thesis, J. Law, University of Wisconsin, August 1962.
9. Digital and Sampled-Data Control Systems, J. T. Tou, McGraw-Hill Book Co., New York, 1959.
10. Control of Chemical Processes by Parameter Perturbation, J. R. Hurley, Ph.D. Thesis, University of Wisconsin, 1963.

DAS — A DIGITAL ANALOG SIMULATOR

R. A. Gaskill, J. W. Harris* and A. L. McKnight***

SUMMARY

Digital Analog Simulation (DAS) is a programming technique which makes a digital computer operate much like an analog computer. The application of the technique to programming an IBM 7090 computer is described. Although it is not intended primarily as an analog computer simulator, the similarity of DAS programming to analog programming is readily apparent. DAS combines remarkable ease and speed of programming with reasonably high computing speed.

The DAS input language is designed to permit a simple and concise description of an analog-style block diagram of the problem to be solved. The blocks in the diagram are restricted to summers, integrators, multipliers, limiters, relays and other "components" for which macro-instructions appear in the DAS compiler. The construction of a DAS block diagram is more straight-forward than an analog computer block diagram because there is virtually no restriction on the number of available components, and because no amplitude scaling is required.

The standard output format provides a complete record of the problem by printing the entire input program and all input data. This is followed by a multiple column table of sequential values of the desired output quantities. These columns of data serve the same purpose as the recording channels of a conventional analog computer output device in that each column provides a complete history of one variable.

INTRODUCTION

A new programming technique, Digital Analog Simulation (DAS), has been developed which makes a digital computer operate much like an analog computer. The technique was developed because it was believed that the key to easy programming is the block diagram approach normally associated with analog computation. Experience with DAS programming for an IBM 7090 computer has emphatically confirmed this belief.

Analog computers contain a set of components which serve as mathematical building blocks. It is natural to prepare a problem for analog computation by drawing a diagram indicating how these blocks should be interconnected. The conversion of a problem statement into block diagram form is fundamentally straightforward. This characteristic of block diagram problem formulation makes analog programming relatively easy. Analog programming would be still easier if it were not necessary to consider limitations of the computer while developing the block diagram. These limitations include: (1) a fixed number of available components of any given kind, and (2) a limited allowable (voltage) magnitude at the output of each component. Such considerations complicate the preparation of a suitable analog computer diagram to the point where it may be argued that it would be just as easy to prepare the problem for digital computation, using a problem-oriented input language such as FORTRAN. But block diagram problem formulation has intrinsic advantages often outweighing

* Martin Company, Orlando, Florida.

** Thompson Ramo Wooldridge, Inc., RW Division, Fort Huachuca, Ariz. Mr. McKnight's participation in this paper commenced prior to his association with TRW.

the difficulties which may be encountered in preparing or implementing the diagram.

The advantages of block diagram problem formulation are particularly evident in the solution of equations representing a mathematical model or simulation of a physical system. Because each block in the diagram corresponds to some component or characteristic of the system under study, the inputs and outputs of each block have physical significance. By recording the outputs of the computer representation of these blocks an extremely thorough picture of the inner workings of the simulated system is obtained. The close association between components of the computer and components of the system under study also permits representing the addition, subtraction, or modification of components in the physical system by analogous changes in the computer.

The objective of the DAS programming technique for digital computers is to profit fully from the many advantages of block diagram problem formulation. The DAS program makes a digital computer appear to contain an indefinite number of integrators, multipliers, summers, function generators and other components normally associated with analog computers. The DAS input program is essentially a description of how these components should be interconnected to solve the problem. In this sense DAS is similar to DEPI[1] and DYSAC[2, 3], but DAS uses a more basic set of components and a simpler procedure for writing the input program. Of course, the components are not physically interconnected as they would be in an analog computer, and it is in the rapid assembly of machine instructions to represent the interconnection of components that DAS is particularly unique.

DAS COMPONENTS

A catalog of DAS components is shown in Table I. Available quantities of the various components are unspecified because the DAS compiler generates as many of each component as required for the problem at hand. Each component (block) is identified by an abbreviated name and a number. The numbers are arbitrarily assigned by the programmer when he prepares the block diagram of his problem and they serve only to distinguish between otherwise identical components.

Each component is represented by an open subroutine (macro-instruction) in the DAS compiler. So far as the programmer is concerned, however, a component is simply a block with one or more inputs and outputs. For example, assume an integrator receives an input from the output of a summer. The output of the integrator is the integral of its input and hence the integral of a sum. The various input-output relationships are indicated in Table I. Constants, K, and initial conditions, IC, do not receive inputs from the other components. Instead, their inputs come from data cards. Up to six numbers can be punched in each data card. The significance of each number of a data card is explained by the corresponding K or IC card. The K and IC components do not require serial numbers because they are completely identified by their inputs. The arbitrary function component requires two kinds of inputs: an independent variable from another component and a table of values from data cards.

DAS PROGRAMMING TECHNIQUE

The DAS programming technique is best explained by an example. Consider the following differential equation:

$$\frac{d^2X}{dt^2} + AX = 0$$

The first step in the preparation of a DAS program is to solve for the highest derivative. This is usually just a matter of writing the original equation in a slightly different form. For the example we have:

$$\frac{d^2X}{dt^2} = -AX$$

The next step is to draw a block diagram of the problem, starting with the assumption that the highest derivative is available as an input to an integrator (Fig. 1). The output of this integrator is the next highest derivative. This derivative is used as the input to another integrator, and the chain of integrators is extended in this manner until the output of the last integrator is the dependent variable, X. It is now a simple matter to complete the block diagram showing how the outputs of the integrators must be combined to produce the highest deriva-

TABLE I

Catalog of DAS Components (Blocks)

Function	Symbol*	Inputs	Outputs
Integrate	In	A	Initial output $+ \int_o^t A dt$
Sum	Sn	$A_i (i = 1, \ldots, K \leqslant 10)$	$\sum_{i=1}^{K} A_i$
Change Sign	NEGn	A	$-A$
Multiply	Mn	A, B	$A \cdot B$
Divide	Dn	A, B	A/B
Constant	K	Data card	Up to six constants
Initial Condition	IC	Data card	Initial values for up to six integrators
Time	IT	Computing Increment (DELT)	Present value of independent variable (time)
Read-Out	ROn	A constant and up to six component outputs	Prints component outputs at constant intervals
Stop Problem	FINn	A, B	Ends computation when $A \geqslant B$
Square Root	SQn	A	$\sqrt{A}$
Exponential	En	A	${}_eA$
Logarithm	LNn	A	$\log {}_eA$
Arc Tangent	ATn	A	$\arctan A$
Sine-Cosine	RESn	A	B output $= \sin A$ C output $= \cos A$
Limit	Ln	A, B, C	$A, B \leqslant A \leqslant C$ $B, A < B$ $C, A > C$
Input Switching Relay	IRn	A, B, C	$A, C > O$ $B, C < O$
Output Switching Relay	ORn	A, B	C output $= O$ D output $= A$ } $B > O$ C output $= A$ D output $= O$ } $B < O$
Dead Space	DSn	A, B, C	$O, B \leqslant A \leqslant C$ $A - C, A > C$ $A - B, A < B$
Time Delay	TDn	A, IT, C	A delayed by C seconds
Bang-Bang	BBn	A, B	$+B, A > O$ $-B, A < O$
Arbitrary Function	Fn	A, Data Cards	Function of A as tabulated on data cards

* The "n" indicates where an identifying number should appear.

tive. In the example the highest derivative is formed by multiplying the output of the last integrator by a constant, $-A$.

The DAS input program is a description of the block diagram, together with a specification of the constants, initial conditions, desired outputs, and conditions for ending computation. Assume we wish to obtain a printed record of the integrator and multiplier outputs as functions of time until time equals $T1$ seconds. The DAS input program may be written virtually by inspection. It looks like this:

Component	*Inputs to Components*
K	C1, C2, T1
IC	I1, I2
M1	I2, C1
I1	M1
I2	I1
RO1	C2, IT, I1, I2, M1
FIN1	IT, T1
END	

The symbol $C1$ takes the place of $-A$ in the program because all symbols must begin with a letter. Each line of the program is called a program statement, and each statement is punched on a separate card. The cards are presented to the computer in the order shown. The first statement indicates that three numbers must be obtained from a data card. The data cards follow the program, and the first data card contains the numbers required by the first statement. These numbers appear on the data card in the same order as the corresponding symbols on the statement card. For example, the third number on the first data card is $T1$, the maximum value of the independent variable (time) to be considered. Floating point notation is used for all data.

The second card of the example program indicates that the initial value for the output of each integrator must be obtained from a data card. The second data card must contain the desired initial conditions in the order listed on the IC statement. The next three statements describe the block diagram. Each statement describes the connections to one component. The component is identified in the left-hand column and its inputs are listed in the opposite column. For best accuracy, the part of the diagram which provides the highest derivative should be described first. The components should be listed in the order indicated by the arrows on the block diagram. In this case, the highest derivative is formed by Multiplier 1. Multiplier 1 receives its inputs from Integrator 1 and $C1$, a constant equal to $-A$. The description of this part of the diagram is contained in the third statement. Following the arrows, the diagram description is completed by writing statements for Integrator 1 and Integrator 2, in that order.

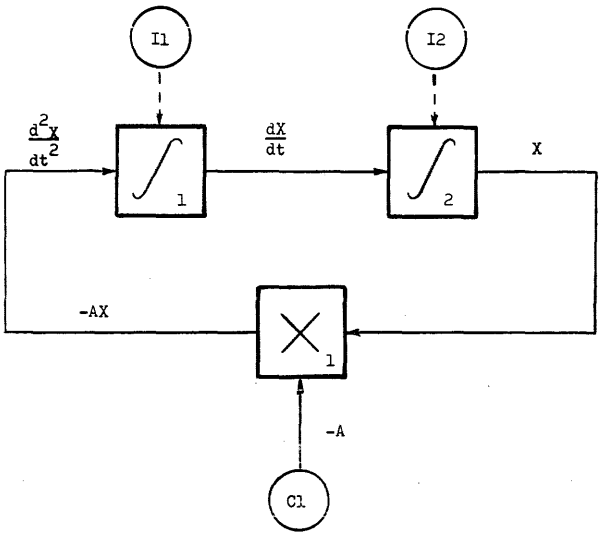


Figure 1. Block Diagram of Example Problem.

The next statement describes what answers are to be printed out and how often. The Read-Out component corresponds roughly to a multichannel strip-chart recorder. The first symbol in the right-hand column specifies the printout interval (analogous to chart speed) and the remaining symbols identify the components which are to have their outputs printed and the order in which they are to appear. For the example, the left-hand column will be time, printed at intervals of $C2$ seconds. The outputs of $I1$, $I2$, and $M1$ will be printed at these times in columns two through four. Two more columns are available but are not needed for this problem. The DAS print-out format is described more completely in a later section. The next to last statement directs the computer to stop computing when the independent variable reaches $T1$ seconds, and the END statement is used to separate the program cards from the data cards.

PROGRAM PROCESSING

When program cards are punched, the keypunch operator inserts an equal sign between the component name and the list of its inputs. The card for the third statement of the example will be:

$$M1 = I2, C1$$

The equal sign is an arbitrary symbol used to assist the computer in distinguishing the component from its inputs. The DAS compiler interprets the input program statements and assembles them into a suitable machine program. Because the present DAS is designed for use with an IBM 7090 computer, it is appropri-

ate to make use of some special features of FAP in the required assembly. One of these features is the provision for writing and using macro-instructions. The DAS compiler contains one macro-instruction for each of the 22 different component types. Each macro-instruction serves as a prototype for a given kind of component. The multiplier macro-instruction is as follows:

```
MULT MACRO      IN1, IN2, OUT
     LDQ        IN1
     FMP        IN2
     STO        OUT
MULT END
```

When the input program statements are translated into the corresponding macro form, the name of the component appears in the OUT location. For example, the statement for Multiplier 1 becomes:

MULT I2, C1, M1

The macro-instructions effectively construct as many of each component as is required by the input program. There is a limit to the total number of components, and for the IBM 7090 this is about three thousand. A separate memory location is reserved for the output of each component listed in the input program so that the outputs of all components are available for print-out or as inputs to other components. By using the same name for the output of a component as for the component itself the writing of an input program is kept extremely simple. When all of the input program statements have been converted to macro-instructions and memory locations have been assigned, the DAS program automatically turns control over to the FAP assembly program and the assembled program is automatically executed. The machine time required prior to execution is about one minute.

DAS METHOD OF SOLUTION

The output of each component in the list of input program statements is updated in sequence by one time increment until all of the outputs have been updated. The time increment (DELT) is automatically 10^{-4} seconds unless otherwise specified by a *K* statement. The value of 10^{-4} seconds was chosen for DELT because it was found to provide the best accuracy for a test problem. The sequence of updating steps is repeated until the solution has been completed. Time (the independent variable) does not enter directly into any of the updating except for the integrators. Rectangular integration is used. The updated integral is obtained by multiplying the integrator input (the integrand) by DELT and adding this product to the present integral. The time required to update all of the components in the list of input program statements may be estimated by multiplying the number of components by 50 microseconds.

DEMONSTRATION PROBLEM

The previous example problem has been expanded to better demonstrate certain features of DAS (Figs. 2, 3, and 4). The differential equation used for the example has a solution which is known to be sinusoidal. For the demonstration problem, *A* has been chosen to provide a solution frequency of 1000/360 cycles per second or one degree per millisecond. The solution is printed at intervals of fifteen milliseconds (fifteen degrees). The initial conditions for *I*1 and *I*2 are 0 and 1, respectively. This

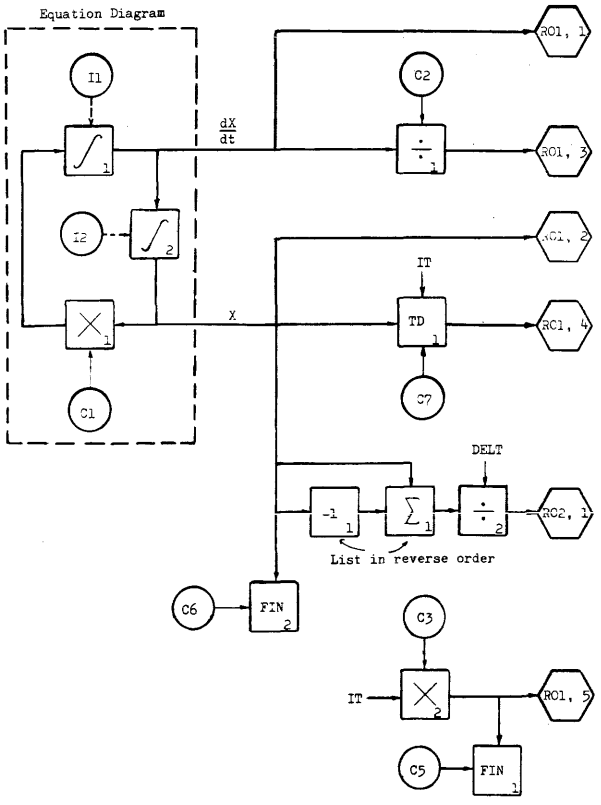


Figure 2. Block Diagram of Demonstration Problem.

provides a cosine output from *I*2 which may be checked against standard tables. The output of *I*1 is divided by its amplitude of oscillation to provide a similar sine output. This output is accurate to five significant figures. The cosine output is passed through a delay component set for a delay of ninety degrees. Ideally, when the output of the delay component appears it should be the same as the sine output. The small error seen in the printed record (Fig. 4) is due to a small error in the cosine. The cosine output is also passed through a differentiator. The output of the differentiator should be (and is) the same as the output of *I*1.

The differentiator consists of three blocks. It would have been a simple matter to include a differentiator component in the catalog of DAS components, but it was decided not to include seldom needed components in the catalog if they could be constructed from the available components. The differentiator takes the first difference of its input and divides by DELT. The first difference is obtained by listing NEG1 out of order. Any component introduces a delay of DELT if it is not listed in the order indicated by the arrows on the block diagram. For ease of comparison, the output of the differentiator is printed directly below the output of *I*1.

Two FIN statements are used. The first one stops computation at 180 degrees if everything proceeds properly. The second stops computation in the event that some error causes the cosine output to significantly exceed unity.

```
    K = C1, C2, C3, C4, C5, C6
    K = C7
   IC = I1, I2
   M1 = I2, C1
   I1 = M1
   I2 = I1
   D1 = I1, C2
   S1 = I2, NEG1
 NEG1 = I2
   D2 = S1, DELT
   M2 = IT, C3
  TD1 = I2, IT, C7
  RO1 = C4, I1, I2, D1, TD1, M2
  RO2 = C4, D2
  RO3 = C4
 FIN1 = M2, C5
 FIN2 = I2, C6
  END =
```

Figure 3. Input Program for Demonstration Problem

The DAS print-out always begins by listing the input program (Fig. 3). This is followed by some program processing information and finally by a table of numbers (Fig. 4) containing all input data as well as the desired output quantities. The significance of the numbers in this table is determined by reference to the input program. The numbers on the data cards are printed first. For the demonstration problem there are three lines of data printed, corresponding to the numbers required by the first three input program statements. All data is printed in floating point notation. The two digit number after the *E* indicates the number of digits that the decimal point should be shifted.

The significance of the numbers following the data is determined by the Read-Out statements. Three Read-Out statements are used for the demonstration problem, all of them calling for print-out at intervals of *C*4 seconds. The first statement calls for printing the outputs of *I*1, *I*2 (the cosine output), *D*1 (the sine output), *TD*1 (the delayed cosine), and *M*2 (the angle) across the page in the order listed. The second statement calls for printing the output of *D*2 directly below the output of *I*1. The third statement provides a double space to separate blocks of answers corresponding to different times.

SPECIAL OPERATIONS

The catalog of DAS components contains relays, limiters, and other special devices for use in solving nonlinear equations or simulating complex control systems. In addition to more obvious aplications, these components serve as basic building blocks for providing other special operations. Three of these special operations will be described. In accordance with analog computer practice, these combinations of components are called circuits.

Absolute Value Circuit

An absolute value circuit is shown in Figure 5. It consists of an input switching relay and a sign changer. The arm of the relay switches to the opposite input whenever X is negative. Because of the sign changer this input is positive when X is negative, so the output of the relay is always positive and equal in magnitude to X.

−3.0462E 02	−1.7453E 01	1.0000E 03	1.5000E−02	1.8000E 02	1.1000E 00
9.0000E−02	−0.	−0.	−0.	−0.	−0.
0.	1.0000E 00	−0.	−0.	−0.	−0.
−4.5172E 00	9.6570E−01	2.5882E−01	0.	1.5000E 01	
−4.5173E 00					
−8.7266E 00	8.6559E−01	5.0000E−01	0.	3.0000E 01	
−8.7266E 00					
−1.2341E 01	7.0649E−01	7.0711E−01	0.	4.5000E 01	
−1.2341E 01					
−1.5115E 01	4.9924E−01	8.6602E−01	0.	6.0000E 01	
−1.5115E 01					
−1.6859E 01	2.5798E−01	9.6592E−01	0.	7.5000E 01	
−1.6859E 01					
−1.7453E 01	−8.7271E−04	10.0000E−01	10.0000E−01	9.0000E 01	
−1.7453E 01					
−1.6858E 01	−2.5966E−01	9.6592E−01	9.6570E−01	1.0500E 02	
−1.6858E 01					
−1.5115E 01	−5.0075E−01	8.6602E−01	8.6559E−01	1.2000E 02	
−1.5115E 01					
−1.2341E 01	−7.0772E−01	7.0710E−01	7.0649E−01	1.3500E 02	
−1.2341E 01					
−8.7266E 00	−8.6645E−01	5.0000E−01	4.9925E−01	1.5000E 02	
−8.7266E 00					
−4.5172E 00	−9.6614E−01	2.5882E−01	2.5798E−01	1.6500E 02	
−4.5171E 00					
4.5593E−06	−9.9999E−01	−2.6123E−07	−8.7003E−04	1.8000E 02	
7.4506E−05					

Figure 4. Solution Print-Out for Demonstration Problem

The circuit of Figure 5 is directly analogous to absolute value circuits used in analog computers. A simpler circuit for obtaining the absolute value of X consists of a single bang-bang component using X for both the A and B inputs.

Sample and Hold Circuit

The sample and hold circuit, shown in Figure 6, is used in the simulation of sampled-data control systems. The input to the sample and hold circuit represents a continuous signal. This signal is sampled periodically, and the latest sampled value appears at the output. It is simply an input switching relay with the output connected to the A input and with the C input coming from a periodic negative trigger pulse generator (to be described next). Between samples, the circuit from A to D and back to A functions as a circulating memory, and the output remains constant. When a negative trigger occurs the output is momentarily connected to the B input. The arm then returns to its normal position and this sampled value of the B input is held in memory until the next trigger pulse occurs.

Trigger Pulse Generator

One trigger pulse generator can operate as many sample and hold circuits as desired. The circuit is shown in Figure 7. Initially, the output of Summer 1 is equal to $-T1$ and the other two outputs are zero. When time (IT) becomes equal to the desired time ($T1$) for the first pulse, then the relay switches to the other input, and the relay output becomes $-P$. This number

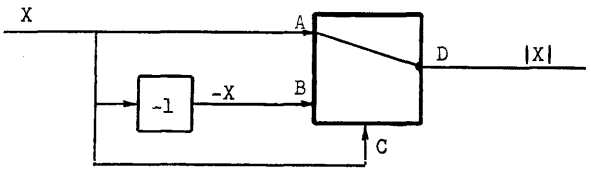


Figure 5. Absolute Value Circuit.

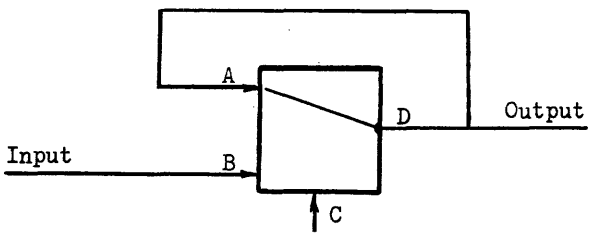


Figure 6. Sample and Hold Circuit.

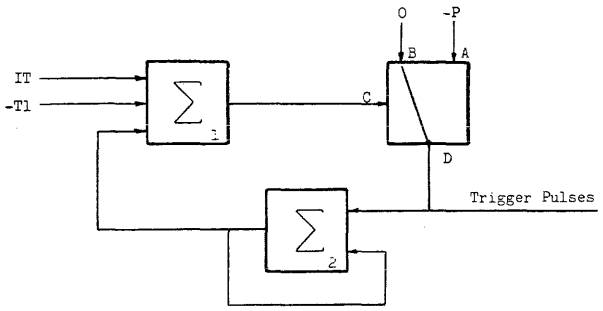


Figure 7. Trigger Pulse Generator.

is added to the present output of Summer 2, making the new output of $S2$ equal to $-P$. This negative jump in the output of $S2$ makes the output of $S1$ also jump to $-P$, causing the relay to return to its original position and the output of the relay to return to zero. The feedback loop around $S2$ causes the output of $S2$ to remain at $-P$.

When IT increases by P seconds, the output of $S1$ again goes positive, making the relay produce another negative pulse. This pulse is added to the present output of $S2$, making the new output equal to $-2P$. This again causes the output of $S1$ to jump to $-P$ and the process is repeated, producing a pulse every P seconds.

SUMMARY OF DAS FEATURES

The outstanding feature of the DAS technique is the extreme ease of programming. Complex problems may be programmed and solved in the same day that they are conceived. No amplitude scaling or knowledge of numerical analysis is required. The usual features of digital computation are retained, such as:

1. Complete printed record of problem and solution.
2. Automatic checking for careless errors in programming.
3. Convenient program storage
4. Rapid access to the computer
5. Extreme dynamic range
6. Good accuracy
7. No drift

Many of the best features of analog computation are obtained, such as:

1. Easy program alteration. Adding or subtracting a card in a DAS program corresponds to changing one or more patching connections in an analog setup.
2. Enhanced problem understanding. The block diagram approach to programming provides valuable problem insight.
3. Outputs of all components available for recording. For problem check-out purposes, it is convenient to print the output of every component. When the minor outputs are no longer needed, the cards which call for their print-out are simply thrown away.

FUTURE MODIFICATIONS

Although DAS is highly useful in its present form, several improvements are being planned. The most important is an approved integrator permitting the use of longer computing increments (DELT) without sacrificing accuracy. This modification will increase the speed of solution and make it economically feasible to solve larger problems. Other planned improvements will provide conveniences such as better labeling of the printed answers, provision for automatic plotting of the answers, and more complete automatic error analysis. With or without these improvements, the DAS programming technique is a significant new tool for the solution of problems in dynamic analysis.

REFERENCES

1. LESH, F., "Methods of Simulating a Differential Analyzer on a Digital Computer," *Journal of the Association for Computing Machinery*, July, 1958.
2. HURLEY, J., "Digital Simulation I: DYSAC, A Digitally Simulated Analog Computer," presented at the AIEE Summer General Meeting, Denver, Colo., June, 1962; University of Wisconsin Reprint No. 548.
3. SKILES, J. and HURLEY J., "Digital Simulation II: Applications," presented at the AIEE Summer General Meeting, Denver, Colorado, June, 1962; University of Wisconsin Reprint No. 549.

SIX DEGREE-OF-FREEDOM SIMULATION OF A MANNED ORBITAL DOCKING SYSTEM

J. C. Fox and T. G. Windeknecht
Control Systems Department
Space Technology Laboratories
One Space Park,
Redondo Beach, California

INTRODUCTION

This paper describes an analog computer simulation of a manned orbital docking system conducted as a feasibility study. In addition to determining the possibility of a human operator performing guidance and attitude control functions with a minimum amount of displayed information, additional purposes of the study were to determine (1) terminal values of range rate, line-of-sight (LOS) angle, LOS rate, lateral miss, relative attitude angle, and fuel consumption, (2) pilot procedures and capabilities and, (3) minimum display requirements.

The orbital docking system studied consisted of an orbiting Earth satellite (target vehicle) and a manned astrovehicle (chaser vehicle). The target vehicle was assumed to be attitude-stabilized in a circular orbit so that the longitudinal (roll) axis was coincident with the orbital velocity vector. Initially, the chaser vehicle was placed ahead of the target in a nearby orbit with a fixed closing velocity relative to the target. Using a periscope view of the target plus radar-derived range and range rate information, the pilot's task was to guide the chaser vehicle toward the target such as to satisfy prescribed terminal conditions. In particular, he was required (i) to maneuver his vehicle into the orbit plane of the target, (ii) to align his roll axis with that of the target, (iii) to establish a specified roll attitude relative to the target, and in this orientation (iv) to reduce the vehicles' separation distance with a decreasing rate of approach. The feasibility of this task was heavily dependent upon the pilots' ability to operate effectively with only the relatively small amount of explicit information provided by the periscope and radar.

The study was oriented such that the initial and final conditions could be related to conditions described in previous studies.[1, 2] In reference 1, the Manned Rendezvous Simulation (MRS) study results were partially utilized to define reasonable initial conditions of chaser vehicle velocity and position with respect to the target vehicle.

The MRS study considered the planar analysis of a rendezvous system consisting of a cooperative orbiting target vehicle and a manned chaser vehicle. The MRS chaser vehicle was assumed to have perfect attitude control such that one body axis was directed along the LOS to the target and a second was aligned normal to the orbit plane. Hence only translational accelerations along and normal to the LOS were considered in the MRS study. In addition, the target vehicle was assumed to continually point its docking face toward the incoming chaser vehicle. Pilots, in the MRS study were provided with metered displays of the following quantities:

(1, 2) Superscript numbers refer to references.

Range and range rate
LOS rate
Lateral velocity normal to LOS
Target elevation with respect to a local horizontal

In the MRS study the initial and terminal values of range were, respectively, 200,000 feet and 1250 feet. Rendezvous was consistently achieved with closing range rates of about five feet per second and LOS rates near one milliradian per second at the terminal range.

The present docking study expanded on the MRS two degrees-of-freedom study by considering the complete six degrees-of-freedom of the dynamics, by assuming a non-cooperating target, and by requiring complete pilot control over all chaser vehicle motion. The target vehicle was assumed to be attitude stabilized about its pitch, yaw, and roll axes with motion restricted to its orbital plane at a constant orbital rate. The target was assumed incapable of guidance accelerations and variations from a 300-mile altitude circular orbit were not considered. It was intended that the docking task should be performed by the pilot using a minimum of displayed information, i.e., in this study, the pilot was provided with the following information:

Optical view of the target vehicle
Range rate and range (on log scale)

The present study considered the chaser vehicle to be initially 3000 feet away from the target vehicle with a closing range rate of 30 feet per second and with negligible* LOS rates (2 milliradians per second or less). Various combinations of 10 degree chaser attitude errors were also included as initial conditions. Termination of each flight occurred when the docking faces of the two vehicles reached an 8 foot separation.

Reference 2 defines the acceptable limits of terminal errors in position and velocity resulting in successful dynamical operation of one docking mechanism considered. The acceptable limits, specified at a vehicle interface range of 10 feet, are

(1) 15 degrees LOS angle (angle between the vehicle roll axes)
(2) 1 deg/sec LOS rate
(3) 1 ft/sec closing velocity
(4) 5 degrees roll error
(5) 2 feet lateral center-to-center miss

The above set of terminal errors was taken to represent a relative minimum performance criteria for the simulated flights.

A fixed-base cockpit simulator was utilized as an integral part of the analog computer setup. Two (three degrees-of-freedom) control sticks were provided in the cockpit such that the pilot commanded on-off translational accelerations along each body axis and proportional attitude rates about each body axis. In addition, the cockpit contained a viewing port which presented the pilot with the simulated optical view of the target on a viewing screen (oscilloscope). Also present on the viewing screen were metered indications of range and range rate.

System Model and Coordinates

The initial system orientation and the geometrical shapes of the vehicles are shown in Figure 1. The cylindrical target dimensions were 24 feet in length and 5 feet in diameter. Referring to Figure 2, three docking face markers (such as lamps) were assumed to be mounted on the docking face of the target. The markers were spaced 90 degrees apart so that a relative roll error between the chaser and target could be determined from the pilot's display.

The chaser vehicle was assumed to have a geometrical shape similar to a truncated cone as shown in Figures 1 and 2. An optical sensor, such as a periscope or TV camera, was as-

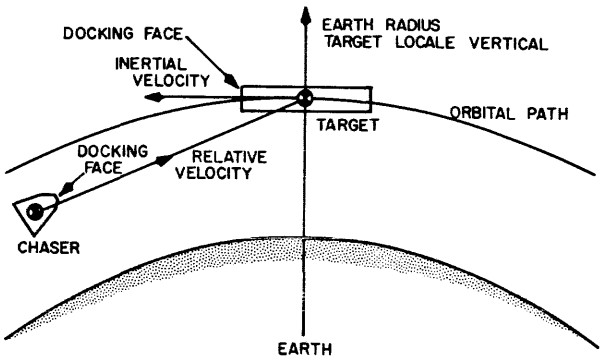


Figure 1. Orbital Docking System Initial Geometry.

* The latter assumption does not appear to limit the study results. This is because in early portions of successful experimental flights, LOS rates of 1 and 2 degrees/second were recorded, indicating the ability of the pilot to overcome such values of LOS rate.

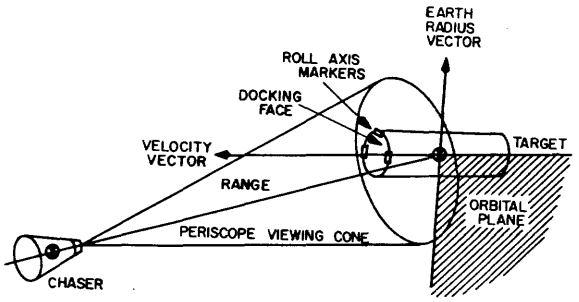


Figure 2. Vehicle Orientation.

sumed mounted on and body-fixed to the chaser docking face with the optical line-of-sight collinear with the roll axis. The chaser translational engines were assumed to be mounted such that (i) misalignments between the line-of-force and center-of-mass could be neglected. The chaser vehicle was also assumed to have (ii) negligible cross products of inertia, (iii) equal principal moments of inertia, and (iv) rotational torques applied as independent couples about each body axis. It is felt that none of these assumptions limit the applicability of the results of the study.* The mass and inertias of the chaser vehicle were treated parametrically, hence numerical values of thrust levels and fuel utilization are given in normalized units.

Equations of motion and display equations for the docking mission were programmed for the computer by utilizing several orthogonal coordinate sets pertaining to both vehicles. One coordinate set is fixed to the target vehicle with origin at the center-of-mass such that the reference directions coincide with the principal axes of inertia of the target. This set is denoted the ($\bar{u}_X$, $\bar{u}_Y$, $\bar{u}_Z$) or target coordinate system. The $\bar{u}_X$ direction is defined as collinear with (but opposite in sign to) the inertial velocity vector of the target vehicle. The $\bar{u}_Y$ direction is defined as collinear with (and directed upward along) the local Earth vertical at the target center-of-mass. Due to the assumed attitude motion of the target, the $\bar{u}_Z$ vector is inertially fixed. The $\bar{u}_X$ and $\bar{u}_Y$ vectors rotate at a constant (orbital) rate about $\bar{u}_z$. (Ω is a positive rate according to the right-hand rule). The chaser vehicle center-of-mass is displaced, respectively, distances X, Y, and Z along the target reference directions $\bar{u}_X$, $\bar{u}_Y$, and $\bar{u}_Z$. This (X, Y, Z) target coordinate system is an expansion of the planar (X, Y) set defined in reference 1.

For the purposes of this study, a set of spherical coordinates (R, α, β) were used to describe the displacement of the chaser center-of-mass relative to the target coordinates instead of the rectangular coordinates (X, Y, Z). In this set, R is the radial distance from the target center-of-mass to the chaser center-of-mass, β is the angle between R and the X-Y plane projection of R, and α is the angle between the X-Y plane projection of R and X-axis. These relationships are illustrated in Figure 3. An explicit relationship between (X, Y, Z) coordinates and the R, α, β) coordinates is given by

$$\begin{aligned} X &= -R\cos\beta\cos\alpha \\ Y &= -R\cos\beta\sin\alpha \\ Z &= R\sin\beta \end{aligned} \qquad (1)$$

To facilitate transformation of chaser body rotations and body rates to the target coordinate system, an "ideal" chaser attitude coordinate system ($\bar{u}_R$, $\bar{u}_S$, $\bar{u}_T$) is defined as shown in Figure 4. (The $\bar{u}_R$, $\bar{u}_S$, $\bar{u}_T$) coordinate system has its origin at the center-of-mass of the chaser vehicle but is not body-fixed. The $\bar{u}_R$ vector is directed from the chaser center-of-

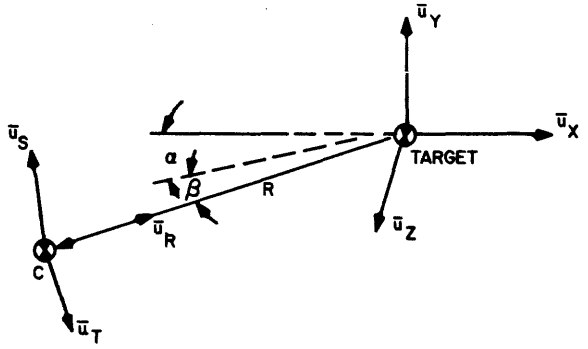


Figure 3. Target Coordinate System.

* The first three effects would all tend to make the attitude control task more difficult for the pilot because of increased coupling between control axes and control functions. However, the attitude control task as given was demonstrated to be trivial even for untrained pilots. Thus, the complication of the task due to the above effects appears negligible. The fourth assumption is valid because attitude control thrust is generally very small compared to the thrust of engines used for guidance.

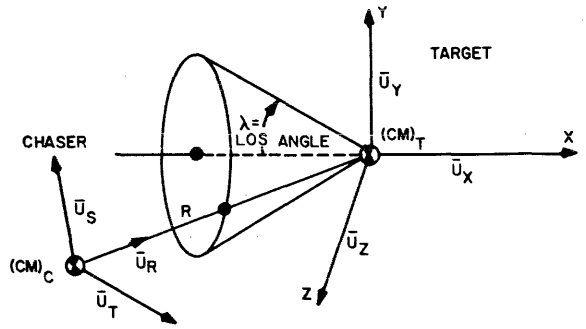


Figure 4. Ideal Chaser Attitude Coordinate System.

mass to the target center-of-mass. $\bar{u}_S$ and $\bar{u}_T$ are chosen such that the set is orthogonal and right-handed. Further conditions used to specify the set are that $\bar{u}_T$ is normal to the local vertical at the target and that it makes an acute angle with the $\bar{u}_Z$ direction. The former requirement specifies the relation $(\bar{u}_T \cdot \bar{u}_Y) = O$. The latter requires that the matrix relating the $(\bar{u}_X, \bar{u}_Y, \bar{u}_Z)$ set with the $(\bar{u}_R, \bar{u}_S, \bar{u}_T)$ set reduce to the unit matrix for $\alpha = \beta = O$. Utilizing these conditions, the (orthogonal) matrix W relating the $(\bar{u}_R, \bar{u}_S, \bar{u}_T)$ set to the $(\bar{u}_X, \bar{u}_Y, \bar{u}_Z)$ set according to

$$\begin{bmatrix} u_R \\ u_S \\ u_T \end{bmatrix} = [W] \begin{bmatrix} u_X \\ u_Y \\ u_Z \end{bmatrix} \tag{2}$$

becomes

$$[W] = \begin{bmatrix} \cos\beta\cos\alpha & \cos\beta\sin\alpha & -\sin\beta \\ \dfrac{-\cos^2\beta\sin\alpha\cos\alpha}{\sigma} & \sigma & \dfrac{\sin\beta\cos\beta\sin\alpha}{\sigma} \\ \dfrac{\sin\beta}{\sigma} & 0 & \dfrac{\cos\beta\cos\alpha}{\sigma} \end{bmatrix} \tag{3}$$

where

$$\sigma \triangleq \sqrt{1 - \cos^2\beta\sin^2\alpha}$$

The line-of-sight (LOS) angle is defined as the angle between $\bar{u}_R$ and $\bar{u}_X$ and here is denoted λ. The LOS angle is related to angles α and β by

$$\lambda = \cos^{-1}(\cos\alpha\cos\beta) \tag{4}$$

By direct differentiation of equation (4), the following expression for λ, the LOS rate, can be obtained:

$$\lambda = \frac{\alpha\sin\alpha\cos\beta + \beta\sin\beta\cos\alpha}{(1 - \cos^2\alpha\cos^2\beta)^{1/2}} \tag{5}$$

A coordinate system, $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$ fixed in the chaser vehicle may be conveniently defined as follows. See Figure 5. The origin of the $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$ set is coincident with the chaser vehicle center-of-mass and the directions of the set

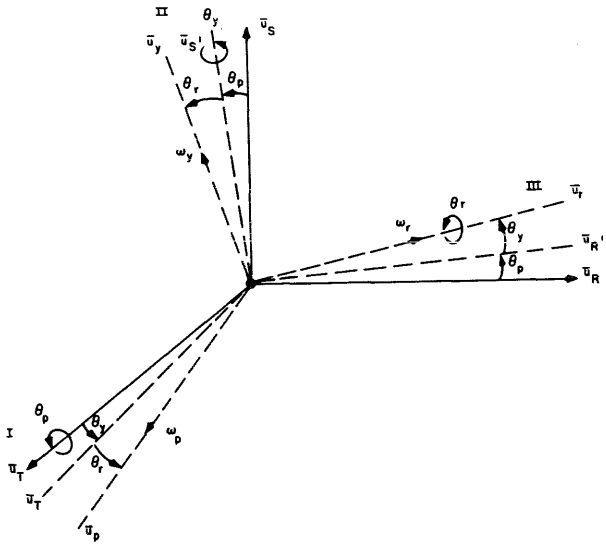


Figure 5. Relation Between the Ideal Chaser Coordinate System $(\bar{u}_R, \bar{u}_S, \bar{u}_T)$ and the Body-Fixed Coordinate Set $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$.

coincide with the principal axes-of-inertia. The $\bar{u}_r$ vector lies along the longitudinal (roll) axis of the chaser and makes an acute angle with $\bar{u}_R$. The $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$ coordinate set is defined relative to the $(\bar{u}_R, \bar{u}_S, \bar{u}_T)$ set of "ideal" attitude coordinates by the sequence of ordered angular rotations depicted in Figure 5. In sequence, the rotations have magnitude θ_p (pitch angle), θ_y (yaw angle), and θ_r (roll angle). The

rotational transformation matrix, $[A]$, which relates the $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$ coordinate system to the $(\bar{u}_R, \bar{u}_S, \bar{u}_T)$ set according to

$$\begin{bmatrix} u_r \\ u_y \\ u_p \end{bmatrix} = [A] \begin{bmatrix} u_R \\ u_S \\ u_T \end{bmatrix} \tag{6}$$

can be shown to be

$$[A] = \begin{bmatrix} 1 & \theta_p & -\theta_y \\ -\theta_p & 1 & \theta_r \\ \theta_y & -\theta_r & 1 \end{bmatrix} \tag{7}$$

under small angle approximations on θ_p, θ_y and θ_r. Note that small angle approximations on this transformation are justified on the basis that the pilot's task was to maintain essentially the "ideal" attitude defined earlier and that an "adequate" control system for this purpose was to be provided him.

In the light of the foregoing definitions, the transformation of coordinates between the target $(\bar{u}_X, \bar{u}_Y, \bar{u}_Z)$ set and the chaser $(\bar{u}_r, \bar{u}_y, \bar{u}_p)$ set may be written as

$$\begin{bmatrix} u_r \\ u_y \\ u_p \end{bmatrix} = [B] \begin{bmatrix} u_X \\ u_y \\ u_Z \end{bmatrix} \tag{8}$$

where

$$[A]\ [W] \triangleq [B] \tag{9}$$

The $[B]$ matrix elements were explicitly generated in the computed simulation and are given by

$$[B] = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \tag{10}$$

where

$$b_{11} = \cos\beta\cos\alpha - \frac{(\theta_p \cos^2\beta \sin\alpha \cos\alpha + \theta_y \sin\beta)}{\sigma}$$

$$b_{12} = \cos\beta \sin\alpha + \sigma\theta_p$$

$$b_{13} = -\sin\beta + \frac{(\theta_p \sin\beta \cos\beta \sin\alpha - \theta_y \cos\beta \cos\alpha)}{\sigma}$$

$$b_{21} = -\theta_p \cos\beta \cos\alpha - \frac{(\cos^2\beta \sin\alpha \cos\alpha - \theta_r \sin\beta)}{\sigma}$$

$$b_{22} = -\theta_p \cos\beta \sin\alpha + \sigma \tag{11}$$

$$b_{23} = -\theta_p \sin\beta + \frac{(\sin\beta \cos\beta \sin\alpha + \theta_r \cos\beta \cos\alpha)}{\sigma}$$

$$b_{31} = \theta_y \cos\beta \cos\alpha + \frac{(\theta_r \cos^2\beta \sin\alpha \cos\alpha + \sin\beta)}{\sigma}$$

$$b_{32} = \theta_y \cos\beta \sin\alpha - \sigma\theta_r$$

$$b_{33} = -\theta_y \sin\beta - \frac{(\theta_r \sin\beta \cos\beta \sin\alpha - \cos\beta \cos\alpha)}{\sigma}$$

The equations of motion for the translation of the chaser c.m. with respect to the target-centered (X, Y, Z) or (R, α, β) coordinate set are well known and defined elsewhere.[2, 3] The equations of motion for body attitude rotations of the chaser vehicle are similary defined in reference 3. For the purposes of this paper, discussion of the derivation of these two sets of motion equations is not considered. The equations of motion implemented in the computer program are set forth in the Appendix.

Displays

In the fixed-base simulator cockpit, the pilot was presented with an oscilloscope display of the docking and rear faces of the target. In addition, log range and range rate markers were superimposed on the same display surface directly below the periscope view as shown in Figure 6. The circular target faces were essentially the same size. The target docking face was identified by roll axis markers which were not present on the rear face. The display dimensions corresponded exactly to a target view as seen through an aperture on the chaser docking face.

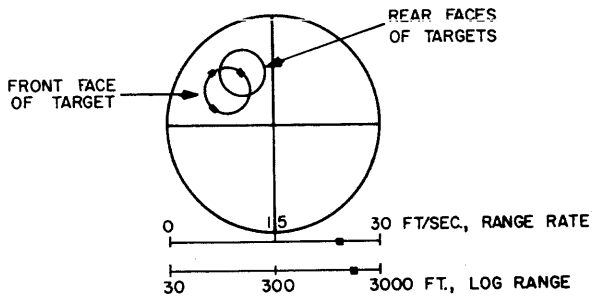


Figure 6. Optical Display.

Again referring to Figure 6, attitude errors were indicated in the display by the distance that the centroid of the two target circles was displaced from the center of the simulated viewing screen. Guidance errors were indicated by the relative displacement of the front and rear circles. This displacement was proportional to the LOS angle in the case of perfect attitude control (i.e., when the centroid of the target coincided with the viewing port center). Errors appeared on the display in this manner because the optical sensor was assumed body-fixed and pointing along the chaser roll axis.

The generation of the periscope image of the target as seen from the center of the chaser docking face may be implemented by considering two closed curves; one representing the (circular) outline of the target docking face and a second representing the outline of the rear face.

Mathematically, the two curves to be displayed may be defined by first describing the components of two general points, one on the rim of each face of the target vehicle, in a particular coordinate system fixed to the chaser vehicle. This coordinate system has its origin at the center of the docking face of the chaser and its directions are parallel to those of the chaser body coordinate system previously defined. The components of these general points required for display are those parallel to the chaser yaw and pitch axes (and are referred to as the chaser vertical and horizontal display components).

The entire outline of each face is described if the two general points previously defined are considered to move around the rim of the target face as a function of time. In particular, one can consider the general points to trace out their respective faces at a constant angular rate. With respect to generating a display of the target, it can be seen that an observer located at the center of the chaser vehicle docking face, who watches the moving points, essentially sees a continuous picture of the faces of the target. A requirement which assures the continuity and accuracy of the image is that the angular rate of the moving points be large compared to the rates associated with the target and chaser vehicle motions.

In an analog computer simulation, the motion of the points on the target faces can be introduced by use of a constant frequency sinusoid generator. The superposition of the target faces on a single oscilloscope display requires use of an electronic switch that commutates the signals which drive the "x" and "y" inputs of the oscilloscope. Roll axis markers may be superimposed on the curve representing the front (docking) face of the target by impulsively increasing the oscilloscope intensity for appropriate values of the phase angle of the output of the sinusoidal generator. These techniques were utilized in the present study.

Referring to Figure 7, a general point (P_f) located on the rim of the target docking face may be expressed in target coordinates as a function of the target cylindrical radius (a) and the distance (l) from the face to the vehicle center of mass:

$$P_f: \begin{cases} X_f = -l \\ Y_f = a \sin \phi \\ Z_f = a \cos \phi \end{cases} \tag{12a}$$

Similarly, a general point (P_r) on the target rear face is,

$$P_r: \begin{cases} X_r = +l \\ Y_r = a \sin \phi \\ Z_r = a \cos \phi \end{cases} \tag{12b}$$

The points P_f and P_r may be conveniently expressed in terms of distances (X_d, Y_d, Z_d) measured along target coordinates but referenced to an origin at the chaser center-of-mass as follows:

$$\left.\begin{aligned} P_f: & \begin{cases} X_{d_f} = -l + R \cos \beta \cos \alpha \\ Y_{d_f} = a \sin \phi + R \cos \beta \sin \alpha \\ Z_{d_f} = a \cos \phi - R \sin \beta \end{cases} \\ P_r: & \begin{cases} X_{d_r} = l + R \cos \beta \cos \alpha \\ Y_{d_r} = a \sin \phi + R \cos \beta \sin \alpha \\ Z_{d_r} = a \cos \phi - R \sin \beta \end{cases} \end{aligned}\right\} \tag{13}$$

The points (P_f, P_r) may next be transformed into chaser body-fixed coordinates by use of the B matrix i.e.,

$$\begin{bmatrix} d_{r_f} \\ d_{y_f} \\ d_{p_f} \end{bmatrix} = [B] \begin{bmatrix} X_{d_f} \\ Y_{d_f} \\ Z_{d_f} \end{bmatrix} \tag{14}$$

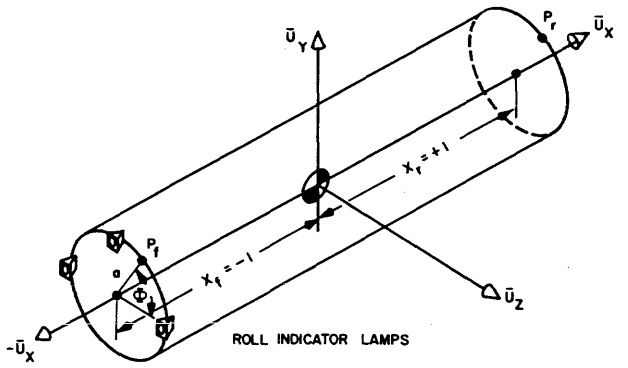


Figure 7. Target Display Surfaces.

The components of P_f and P_r required for display are those which lie along the chaser yaw and pitch axes. These quantities are, d_{y_f} and d_{p_f} respectively, in the case of the front target face. Performing the indicated matrix multiplication gives,

$$\begin{aligned} d_{y_f} &= b_{21}(-l + R\cos\beta\cos\alpha) \\ &\quad + b_{22}(a\sin\phi + R\cos\beta\sin\alpha) \\ &\quad + b_{23}(a\cos\phi - R\sin\beta) \\ d_{p_f} &= b_{31}(-l + R\cos\beta\cos\alpha) \\ &\quad + b_{22}(a\sin\phi + R\cos\beta\sin\alpha) \\ &\quad + b_{33}(a\cos\phi - R\sin\beta) \end{aligned} \tag{15}$$

Identical expressions may be obtained for the rear face components in chaser coordinates except that $(-l)$ is replaced by $(+l)$. Since the b_{ij} terms are functions of the chaser attitude angles, the above equations give a general point on each face of the target in chaser coordinates as a function of range (R), LOS angle (α, β), attitude angles $(\theta_r, \theta_y, \theta_p)$ and target dimensions (a, l).

Equations (15), must be divided by the distance (Q) from the center of the chaser docking face to the center of the front face of the target. This is required in order to maintain the proper ratio of optical image magnitude to optical path length in the quantities to be generated as oscilloscope display inputs. To generate the continuous optical display of the target faces, the quantities $\cos\phi$ and $\sin\phi$ may be replaced with $\cos\omega t$ and $\sin\omega t$. These functions are derived from a sinusoidal generator operating at a constant angular frequency (ω).

In light of the foregoing, the oscilloscope input quantities are defined by:

$$\left.\begin{aligned} f_V &= \frac{d_{y_f}}{Q}, && \text{vertical optical component of target front face in chaser coordinates.} \\ f_H &= \frac{d_{p_f}}{Q}, && \text{horizontal optical component of target front face in chaser coordinates.} \\ r_V &= \frac{d_{y_r}}{Q}, && \text{vertical optical component of target rear face in chaser coordinates.} \\ r_H &= \frac{d_{p_r}}{Q}, && \text{horizontal optical component of target rear face in chaser coordinates.} \end{aligned}\right\} \tag{16}$$

Writing out these expressions results in

$$\left.\begin{aligned} f_V = \frac{d_{y_f}}{Q} &= b_{21}\left(-\frac{l}{Q} + \frac{R}{Q}\cos\beta\cos\alpha\right) \\ &\quad + b_{22}\left(\frac{a}{Q}\sin\omega t + \frac{R}{Q}\cos\beta\sin\alpha\right) \\ &\quad + b_{23}\left(\frac{a}{Q}\cos\omega t - \frac{R}{Q}\sin\beta\right) \\ f_H = \frac{d_{p_f}}{Q} &= b_{31}\left(-\frac{l}{Q} + \frac{R}{Q}\cos\beta\cos\alpha\right) \\ &\quad + b_{32}\left(\frac{a}{Q}\sin\omega t + \frac{R}{Q}\cos\beta\sin\alpha\right) \\ &\quad + b_{33}\left(\frac{a}{Q}\cos\omega t - \frac{R}{Q}\sin\beta\right) \end{aligned}\right\} \tag{17}$$

Expressions for the rear face components (r_V, r_H) are identical to Equations (17) except that $\left(\frac{l}{Q}\right)$ replace $\left(-\frac{l}{Q}\right)$.

In this study (Q), the optical path length, was considered to be given by the approximate expression

$$Q \approx R - (m + l) \tag{18}$$

where m is the distance along the chaser roll axis from the center-of-mass to the chaser periscope aperture. (See Figure 8.)

This assumption is justified on the basis that, (1) at long range, when attitude errors and the LOS angle may be great $(m + l)$ is much less than R, and (2) at short range, because attitude errors and the LOS angle must then be small, the distances Q and R are essentially collinear.

Near the terminal conditions, under the assumption that β and α are small angles, Equations (17) reduce to

$$\left.\begin{aligned} f_V &= -\theta_p\left(-\frac{l}{Q}+\frac{R}{Q}\right)+\left(\frac{a}{Q}\sin\omega t+\frac{R}{Q}\alpha\right)+\theta_r\left(\frac{a}{Q}\cos\omega t\right) \\ f_H &= \theta_y\left(-\frac{l}{Q}+\frac{R}{Q}\right)-\theta_r\left(\frac{a}{Q}\sin\omega t+\frac{R}{Q}\alpha\right)+\left(\frac{a}{Q}\cos\omega t\right) \end{aligned}\right\} \quad (19)$$

Because the angles (θ_p, θ_y, α, and β) must all be small for successful docking, the products $\left(\theta_p\frac{R}{Q}, \theta_y\frac{R}{Q}, \alpha\frac{R}{Q}, \text{ and } \theta_r\,\alpha\frac{R}{Q}\right)$ are all second-order terms as R becomes small. This fact justifies the approximation that for R <300 feet, $\frac{R}{Q}\approx 1$ in these terms. If a reasonable terminal configuration is achieved, the front face display quantities will be dominated by the terms:

$$\left.\begin{aligned} f_V &\approx \theta_p\frac{l}{Q}+\frac{a}{Q}\sin\omega t \\ f_H &\approx -\theta_y\frac{l}{Q}+\frac{a}{Q}\cos\omega t \end{aligned}\right\} \quad (20)$$

Equations (20) present the form of the displacement of the target face centers $\left(\theta\,\frac{l}{Q}\right)$ corresponding to attitude error and the terminal magnitude of the circular docking face radius $\left(\frac{a}{Q}\right)$.

Figure 9 depicts typical errors displayed during a flight (although attitude errors ($e_{\theta y}$, $e_{\theta p}$) would normally be corrected prior to the 500 ft. range shown). In Figure 9, points 1, 2 and 3 were not explicitly displayed and hence were estimated by the pilot. In order to correct the errors displayed, the pilot would: (1) command a negative attitude pitch rate, ω_p, until $e_{\theta p}$ was nulled, (2) command a left yaw rate, ω_y, until $e_{\theta y}$ was nulled, (3) command a clockwise roll rate, ω_r, to correct for θ_r, (4) apply a positive vertical guidance thrust until a rate was detected in e_α, then apply compensating negative thrust to null the rate (α) such that the rate would become zero when e_α was small, and (5) apply positive horizontal guidance thrust to correct e_β in the same manner as in (4).

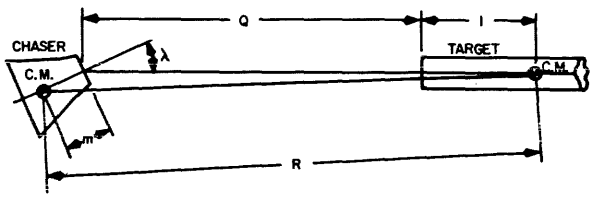


Figure 8. Terminal Geometry.

During the initial portion of a flight, the pilot was provided with a magnified display of the target vehicle. The percent of magnification used increased the optical view of the target diameters by a factor of 20 and corresponded to a periscope viewing cone angle of 3.5 degrees. The display magnification was required to provide adequate LOS and attitude information to the pilot during the time when a normal optical image would be extremely small (i.e., for 3000 to 300 feet range). When the range decreased to 150 feet (nominally) the pilot manually switched to a non-magnified view of the target. This short range display assumed a periscope viewing cone of 70 degrees. The terminal viewing angle was one of the conditions which determined the minimum range considered in the study. The minimum (vehicle) interface range was also dependent upon the target docking face diameter (5 feet) and upon the width on the display screen of the target docking face at termination (½ display screen width). Using the above factors, the terminal interface range was then fixed at 8 feet.

The range and range rate information shown in Figure 9 were assumed to be radar derived. The resolution of visual determination of these quantities was purposely limited by the width of the display markers shown in the figure. The marker widths corresponded to an uncertainty in range rate of ± .25 fps and an uncer-

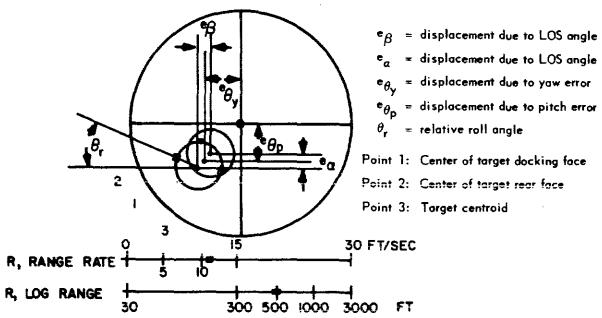


Figure 9. Error Interpretation from Optical Display.

tainty in short scale range of ± 3 feet. For the final phase of the flight (range less than 300 feet), a fine indication of range rate was supplied on a meter with a scale resolution of ± 0.1 feet per second.

The simulation of the target optical image did not include the parallax effect of the front and rear target faces. The parallax effect is significant at short range when the front face appears larger than the rear face. LOS angles less than a value determined by the range cannot then be determined in the physical case. This limitation, however, does not restrict the results of the study (as obtained from use of the simulated display) since LOS information can easily be derived from a view of the targets' front face alone.

A method of obtaining LOS information near the terminal range, with a sensor of the kind as considered in this study, requires addition of an auxiliary LOS angle indicator mounted on the target docking face. As shown in Figure 10 the auxiliary indicator consists of an illuminated collapsible rod on which are mounted LOS centering disks. With this device, LOS angles can still be detected accurately when the rear face view is lost. This LOS indicator could also be mounted on the rim of the target docking face if the chaser periscope was correspondingly rim-mounted. This configuration would also serve to indicate roll errors.

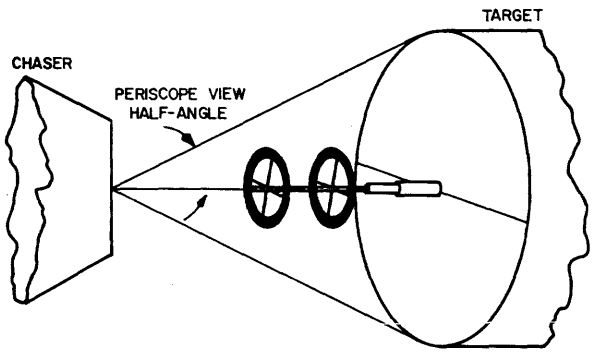


Figure 10. Terminal LOS Indicator.

Controls

As shown in Figure 11, the pilot was provided with two 3-axis controllers for guidance and attitude control corrections. The left-hand controller commanded translational accelerations along the longitudinal (roll), vertical

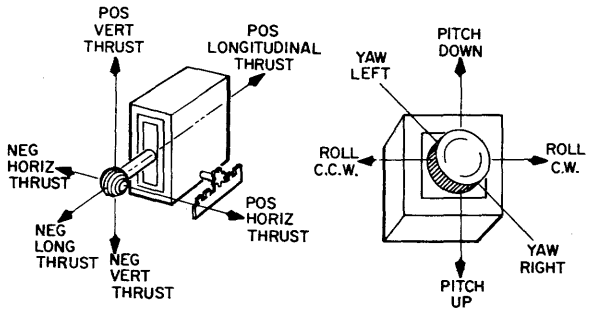


Figure 11. Translational and Rotational Controls.

(yaw) and horizontal (pitch) axes of the chaser. The accelerations generated on the chaser body were in the same direction as the hand motion. As shown in Figure 12, the chaser translational engines were assumed to be the on-off type. Three translational channels (identical except for thrust level) were used for generation of longitudinal, vertical, and horizontal accelerations.

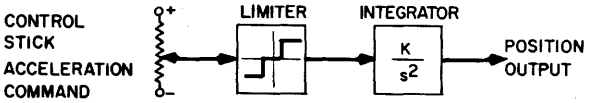


Figure 12. Guidance Command System.

With the right hand controller body angular accelerations could be generated for attitude control. The attitude control channels contained rate gyro feedback as shown in Figure 13. Three identical channels were provided for attitude control about each chaser body axis. The commanded rates were in the same direction as the hand motion, e.g., deflecting the attitude control stick downward produced a negative (nose down) pitch rate.

In addition to the controllers shown in Figure 11, other controls available to the pilot were the long range/short range periscope magnification switch and an on/off switch for initiating and terminating the flight.

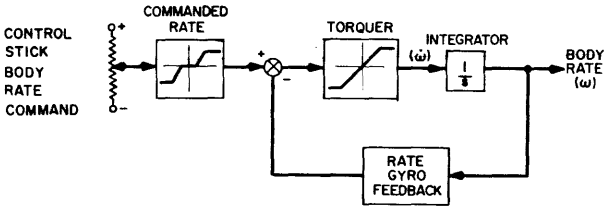


Figure 13. Attitude Control System.

System Performance

Five pilots (referred to as pilots *A* through *E*) were selected for simulator training. Selection was based on the requirement of non-familiarity with the orbital docking system. However, pilots *D* and *E* were deliberately selected because of their previous aircraft flight training and experience. The latter was done for the purpose of comparing aircraft flight procedures with the simulated orbital flight tasks and also to determine if previous aircraft experience provided an advantage in achieving successful docking.

Each pilot received training prior to the data flights from which the statistical averages were determined. The training started with an explanation of (1) the system and its characteristics, (2) desired terminal conditions for a flight, (3) the 3-axis controllers and effect of accelerations, and (4) the optical display, range and range rate markers (including the fine range meter). Subsequently, a total of 2 hours of test flights (prior to making the data flights) were made by each pilot in ½ hour increments. During the test flights, discussion of the pilots' progress took place freely and suggestions for improvement were made.

It was suggested,[3] prior to the start of the simulation, that the pilot follow a nominal approach guidance law determined by keeping the range rate and log range markers vertically aligned (See Figure 6). This procedure, however, resulted in excessive LOS rates and angles during the initial part of the flight. An experimentally determined R-R schedule is shown in Table 1 below.

Range Rate, ft/sec	Range Interval, feet
25	3000-2000
20	2000-1000
10	1000- 500
5	500- 100
2	100- 50
1 or less	50-Dock (28)

Table 1: Approximate Range Rate-Range Schedule

A suggested range of values for guidance and control parameters was given in one of the study directives. Through experimentation, a satisfactory set of values was determined for use in the simulation, all within the suggested limits. The control parameters used during data flights were:

Longitudinal acceleration	1.5 ft/sec^2
Vertical and horizontal acceleration	0.5 ft/sec^2
Command body rate saturation level	1.15 deg/sec
Rotational acceleration saturation level	0.7 deg/sec^2

Note the use of a larger translational acceleration along the longitudinal axis than along the other two body axes. This proved desirable so that large corrections in range rate could be made in a relatively short time.

Each pilot made an identical set of 12 data flights (starting with 12 sets of initial conditions). The sequence of initial conditions was varied randomly for each pilot. The data referred to in the following was derived from the successful flights.

A set of average terminal values has been computed for each pilot (*A* through *E*) so that comparisons of average pilot performance can be made. Further, an overall set of statistical average terminal values has been compiled. The originally specified acceptable limits on terminal values are easily satisfied by the overall average values as given in Table 2.

Terminal LOS angle	4.2 degrees
Terminal LOS angular rate	0.5 degrees/sec
Terminal Lineal closing velocity	0.4 feet/sec
Terminal Relative roll angle	3.3 degrees
Terminal Lateral center-to-center miss	10 inches
Time of Flight	6.7 minutes
Normalized* Guidance Fuel used	71 feet/sec
Normalized* Attitude control fuel used	19.5 degrees/sec

Table 2: Overall Statistical Average Terminal Values (5 pilots)

In Figures 14 through 21 the average terminal values of each pilot are presented on a bar graph to facilitate comparison of individual pilot performances. The overall average is indicated with a dashed line.

* Body mass and inertias not explicitly defined numerically.

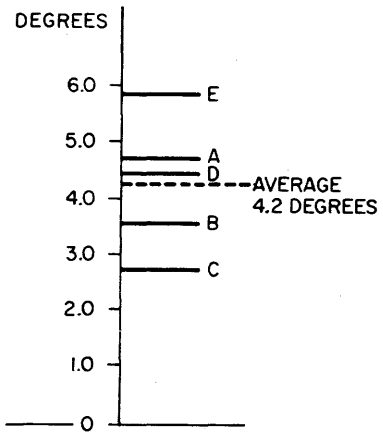


Figure 14. Average Terminal LOS Angle.

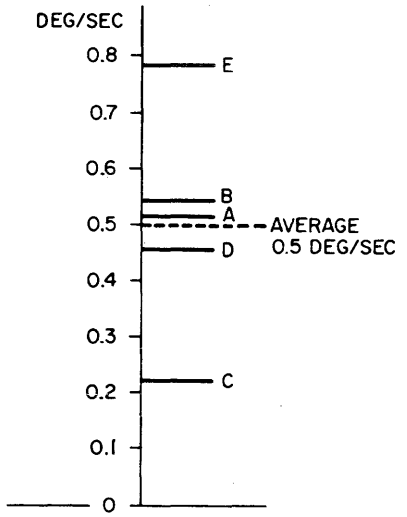


Figure 15. Average Terminal LOS Rate.

Referring to Figures 14 and 15, the variation of the individual average of the pilots from the overall average is approximately 50% of the overall average value for both LOS angle and LOS rate. Note that excluding pilot *E* data would lower the overall average values significantly. On an individual flight basis, the extremes of terminal LOS angle and rate recorded were:

LOS angle	0.5 to 11.0 degrees
LOS rate	0.1 to 1.0 deg/sec

The overall average value of range rate (0.4 ft/sec) and the variation of individual averages from this value are small as shown in Figure 16. The acceptable limit on terminal range rate was defined to be 1 ft/sec. Hence these average terminal values easily satisfy the requirement. A large variation between pilots in lateral miss was recorded (Figure 17). It is felt that this variation would be reduced with increased pilot training.

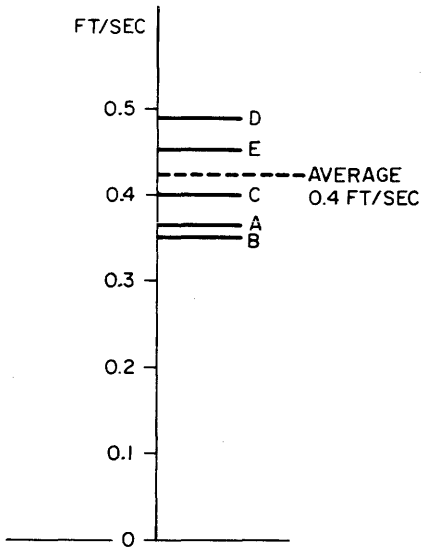


Figure 16. Terminal Range Rate.

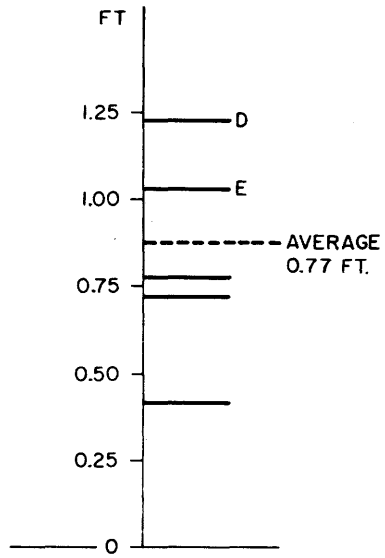


Figure 17. Terminal Lateral Miss.

Average roll errors are shown in Figure 18. It should be stated in connection with Figure 18 that during the earlier flights, some of the pilots concentrated on LOS corrections and neglected to correct the roll error until almost the last possible moment. After being instructed to devote more attention to roll angle error, the corrections were made quite easily. In view of this, it is reasonable to assume that the average roll angle error as shown is actually greater than would be realized with more training.

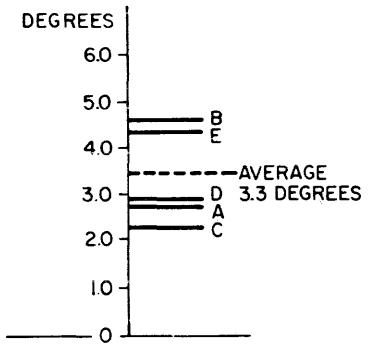


Figure 18. Terminal Roll Angle Error.

As shown in Figure 19, four of the pilots had similar average flight times, while pilot *B*'s average is somewhat lower. Pilot *B*'s data perhaps gives an indication of the minimum flight time to be expected for the given set of initial conditions. Note that pilot *B*'s average terminal conditions are generally below the overall average; hence his performance was not generally degraded relative to the other pilots by maintaining a high closing velocity.

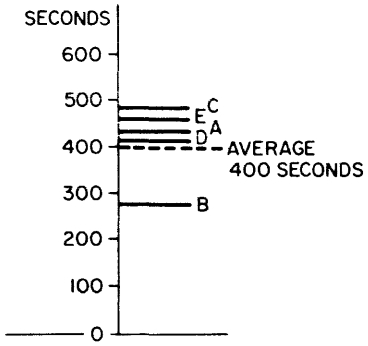


Figure 19. Time of Flight.

As indicated by Figure 20, the individual deviations from the overall average of guidance fuel required was small, indicating that all the pilots encountered approximately the same degree of difficulty in making efficient guidance corrections. Note that an "absolute" minimum of guidance fuel would correspond to the initial condition on range rate of 30 ft/sec. However, orbital coupling in general increases the "absolute" minimum somewhat above this value as does the requirement for maneuvers to terminate in the target's orbital plane. The deviations from the overall average of attitude control fuel in Figure 21 appear to be relatively large. However, if typical values of inertias, moment arms and specific impulse are assumed for the vehicle, the individual averages of normalized ACS fuel as well as the overall average can be shown to be very small in terms of weight.

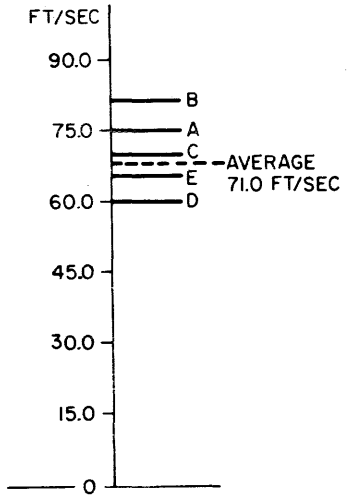


Figure 20. Normalized Guidance Fuel.

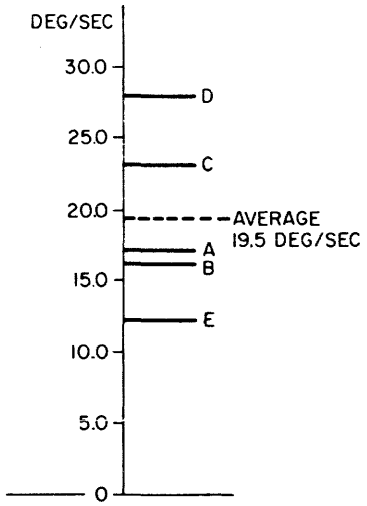


Figure 21. Normalized ACS Fuel.

DISCUSSION

Minimal display requirements are related to the level of pilot training. The display used in this simulation has been demonstrated to be adequate (since docking feasibility was established). It is felt, however, that pilot training was not high enough to permit optimal utilization of the displayed information. One of the most neglected features of the display was the roll error indication. Roll errors frequently were not corrected until very near the terminal

point or not at all. This was because the pilots had not fully integrated left and right hand motions. In the latter part of the data flights, when the training level was higher, the errors due to concentration on one control or the other decreased considerably.

Only approximate indications as to the minimum display requirements were derived from the study due to the limited time available. Pilot *A* made three flights with no R-$\dot{R}$ information given, the first which was very successful. The second flight was moderately successful, while the third flight was not successful. Pilot *D* made one flight with no R-$\dot{R}$ information, but was given a metered indication of LOS angle. This flight was moderately successful. It appears that a highly trained pilot may be able to adequately estimate values of R and $\dot{R}$ from the display alone. It is conceivable that docking systems need provide the pilot with only a periscope display.

A potentially useful addition to a display would be a metered indication of LOS rate. This information could decrease the difficulty of the flight significantly as LOS rate was the quantity most difficult for the pilot to estimate. As the study has shown, however, LOS rate information is not *necessary* for the success of a flight. Of course, hardware considerations may preclude the sensing of LOS rate.

The most sensitive factor affecting the relative success of the data flights was the terminal value of lateral miss. In no case was a flight unsuccessful due to unacceptable terminal values of LOS angle, LOS rate, or roll angle error. Eight flights were unsuccessful because of unacceptable lateral miss. It is felt that some of these large values of lateral miss may have been caused by a contradiction in the pilot's instructions. For example, the pilots were told to align the target centroid with the display center so that the proper guidance corrections could be easily determined. However, a contradiction of this instruction was given, as the pilots were also told to perform attitude corrections such that the target front face would be centered at the terminal point. The pilots were also instructed not to make extreme attitude corrections during the last few seconds of the flight although some of the pilots did not adhere to this procedure. If these last second corrections had been made, the lateral miss could have been corrected at the terminal point in many cases (but not without a subsequent change in LOS angle).

One observation that can be made from the study data is that no particular set of initial conditions proved to be significantly difficult compared to any other set. It can be concluded from this that the relative success of the flight is independent of the initial conditions (within the limits given).

To determine the relative success of a flight, an error criterion was developed as the sum of weighted values of the terminal conditions. By intuitive reasoning, velocity quantities were assigned a squared weight while angles and lengths were weighted linearly.

Nominal terminal values of a successful flight were arbitrarily taken to be:

0.5 ft/sec range rate ($\dot{R}$)
5.0 degree LOS angle (λ)
0.5 degree/sec LOS rate ($\dot{\lambda}$)
5.0 degree rel. roll angle (θ_r)
0.5 ft lateral c-c miss (δ)
70 ft/sec norm. guid. fuel (F_G)
20 degree/sec norm. ACS fuel (F_A)

Constant weighting factors were assigned to make each term in the following error criterion equal to unity for a nominally successful flight.

$$E_T = 4(\dot{R})^2 + \frac{\lambda}{5} + 4(\dot{\lambda})^2 + \frac{\theta_r}{5} + 2(\delta) + \frac{F_G}{70} + \frac{F_A}{20}$$

Defining E_T = overall terminal error, the nominal value is;

$$E_{Tnom} = 4(.5)^2 + \frac{5}{5} + 4(.5)^2 + \frac{5}{5}$$

$$+2(.5) + \frac{70}{70} + \frac{20}{20} = 7$$

As an indication of the relative success of the pilots' orbital docking missions, a tabular account of the four most successful flights made by each pilot is presented below, as determined from the (E_T) error criteria.

Several interesting conclusions may be noted from the selected data. The most successful flight by each pilot (with lowest E_T) was always made on the later flights (sequence No. 6 or later). Only 5 flights of the 20 shown were made with a sequence number lower than 6. The 11th flight for each pilot (none with the same initial conditions) was among his most successful, indicating that increased training level allowed a significant improvement in the later flights. A more detailed analysis and discussion of the data is given in Reference 3.

Pilot	Flight Sequence No.	Initial Conditions, degrees					Flight IC No.	Flight Time, sec.	$E_{T_{nom}}=7.0$ E_T
		α	β	θ_r	θ_y	θ_p			
A	11	+20	0	0	10	10	2	300	3.81
	9	+20	+20	0	10	10	4	423	3.86
	6	20	0	10	0	0	6	464	5.42
	4	−20	0	10	0	0	5	348	7.11
B	11	0	20	10	0	0	7	311	5.03
	4	20	20	10	0	0	8	225	5.06
	2	−20	0	10	0	0	5	238	5.10
	7	+20	0	0	10	10	2	326	6.10
C	8	0	20	0	10	10	3	372	3.27
	6	+20	20	0	10	10	4	437	3.66
	11	+20	0	10	10	10	10	388	3.91
	10	−20	0	0	10	10	1	351	4.95
D	6	20	0	10	0	0	6	260	3.94
	7	−20	0	0	10	10	1	291	5.50
	10	0	20	0	10	10	3	308	5.51
	11	−20	0	10	10	10	9	485	8.99
E	10	0	20	10	0	0	7	342	3.61
	5	−20	0	10	10	10	9	459	5.05
	1	0	20	10	10	10	11	510	6.84
	11	20	20	10	0	0	8	412	7.85

TABLE 3

Successful Flight Data

APPENDIX

The translational equations of motion are derived from perturbation equations[1] where the difference in orbital radius is small compared to the total orbital radius with respect to earth center-of-mass. In the ($\bar{u}_X$, $\bar{u}_Y$, $\bar{u}_Z$) coordinate system, the translational equations are:

$$\left.\begin{aligned} a_X &= \ddot{x} - 2\Omega\dot{y} \\ a_Y &= \ddot{y} + 2\Omega\dot{x} - 3\Omega^2 y \\ a_Z &= \ddot{z} + \Omega^2 z \end{aligned}\right\} \quad \text{(A-1)}$$

As chaser body translational accelerations are generated along body axes in the ($\bar{u}_r$, $\bar{u}_y$, $\bar{u}_p$) set, the $[B]$ matrix transformation is utilized such that;

$$\begin{bmatrix} a_X \\ a_Y \\ a_Z \end{bmatrix} = [B] \begin{bmatrix} a_r \\ a_y \\ a_p \end{bmatrix} \quad \text{(A-2)}$$

where (a_r, a_y, a_p) are pilot-commanded accelerations. The chaser body rotational accelerations are given by;

$$\left.\begin{aligned} \dot{\omega}_r &= \frac{M_r}{I} \\ \dot{\omega}_y &= \frac{M_y}{I} \\ \dot{\omega}_p &= \frac{M_p}{I} \end{aligned}\right\} \quad \text{(A-3)}$$

where M_r M_y, M_p, are, respectively the control torques applied about the roll, yaw and pitch axes.

REFERENCES

1. WAKAMIYA, Y. and WARD, J., "Manned Rendezvous Simulation," STL/TM 9313.8-153, November 1961.
2. WARD, J., and WILLIAMS, H., "Orbital Docking Dynamics", ARS Paper 1953-61. Guidance, Control and Navigation Conference, August 1961.
3. FOX, J., and WINDEKNECHT, T., "Six-Degree-of-Freedom Simulation of a Manned Orbital Docking System", STL/TM 9352.8-37, April 1962.

APPLICATION OF HYBRID ANALOG AND DIGITAL TECHNIQUES IN THE AUTOMATIC MAP COMPILATION SYSTEM

Dr. S. Bertram
Thompson Ramo Wooldridge Inc.
RW Division
8433 Fallbrook Avenue
Canoga Park, California

SUMMARY

The Automatic Map Compilation System, developed and operating at Thompson Ramo Wooldridge abstracts terrain altitude information from aerial photographs by correlating the imagery appearing on stereo pairs, and outputs contour information and new photographs in which the imagery appears in "true" orthographic projection position. The system uses a small digital computer to control analog elements that provide access to the photographic store and process the resulting signals to measure the altitude errors; the errors are then provided as an input to the computer. The system operates through a set of continuous profiling operations to cover the stereo area.

Since the input for the map compilation is in analog form as pairs of photographs, some precision analog equipment is required for the processing. The solution described avoids transferring the information to a digital store, and thus tremendously simplifies the storage and data handling problem. In effect, the integration of the computer with the analog elements produces a special purpose computer that is extremely efficient for the application. The system represents a solution to a problem which until recently was believed to lie exclusively within the domain of human sensory functions, viz., the precision mechanization of stereo perception for the purpose of measuring altitudes from aerial photographs.

INTRODUCTION

The Automatic Map Compilation System developed at Thompson Ramo Wooldridge under the auspices of the U.S. Army Engineer Geodesy, Intelligence and Mapping Research and Development Agency correlates the imagery appearing on stereo pairs of aerial photographs and outputs a chart showing the altitude contour intervals over the stereo area and a new photograph in which the imagery has been moved so as to appear in correct orthographic projection position to a selected scale. The system utilizes a combination of digital and analog techniques to achieve the required accuracy and speed of operation. In effect, a small digital computer is integrated with an analog memory and other peripheral equipment to obtain a very effective special purpose computer.

The input data for a given compilation is in the form of a pair of aerial photographic transparencies together with pertinent camera data; i.e., position and attitude of camera for each transparency, focal length of camera and distortion characteristics of the lens. The amount of detail information available in the photo-

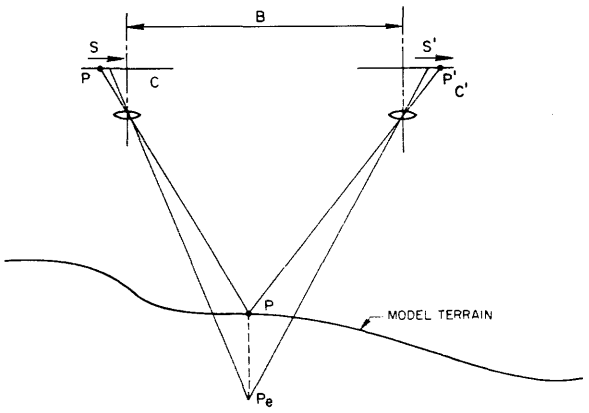
graphs and the nature of the access requirements makes it expedient to use the original photographs as the principal data store for the system.

The system therefore requires a digital computer being on-line with suitable scanning equipment to provide access to the photographic store. It is also convenient to process the photographic data external to the computer and to store the desired output information on new photographs prepared by the equipment. The process then reduces to one in which the digital computer provides precise control information to the analog system with the analog system providing feedback in the form of measured errors which then serves to keep the computer functioning in response to the photographic store.

DEVELOPMENT OF HEIGHT-ERROR SIGNALS FROM STEREO PAIRS

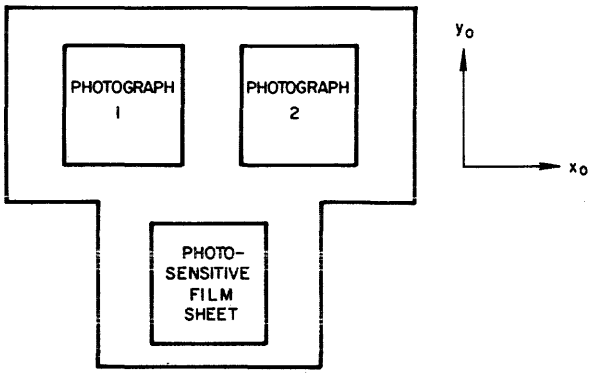
The development of a height-error signal, the key to automatic map compilation, can be seen by reference to Figure 1. Two camera stations C and C' are shown together with an object point P on the surface and its images p and p' in the film plane for the two camera positions. Suppose that the point P was, in some manner, estimated to be at P_e, i.e., below its correct position. One would then be led to look for p and p' by scans s and s' centered on the two representations of P_e; it is obvious that p appears to the left of center and p' to the right of center in these two scans. If P_e had been estimated too high instead of too low, then p would appear to the right of center and p' to the left of center in the two scans. Since P is an arbitrary point in the stereo field, its location in a scan is meaningless. Of extreme significance, however, is a comparison of the position of corresponding imagery within the two scans. If the altitude selected is correct, corresponding imagery will appear at the same positions in the two scans, while if there is any altitude error it will show up as a proportionate shift in the positions in the two scans. A measurement of this shift is, therefore, equivalent to a direct measurement of the error in the estimate of altitude. It is of interest to note that the camera separation B is made large—of the order of the altitude—to exaggerate the shifting of the images with altitude changes.

In the Automatic Map Compilation System, flying-spot scanners are used to obtain signals from areas estimated to be centered on the same point on the surface. If, as shown in Figure 1, scanning proceeds from left to right a low estimate of altitude will yield signals in which elements from C are ahead in time of corresponding elements from C'; for a high height estimate, the reverse is true. Correlation circuitry, described below, is used to match corresponding signal components and to provide an appropriate output to the related control elements.



SYSTEM DESCRIPTION

The photographs to be compiled, in the form of positive transparencies, are mounted together with a photosensitive film sheet on a common carriage, as shown in Figure 2. The carriage is mounted on precision ways, and arrangements made to communicate its position to the computer. For the purpose of the compilation, a rectangular system of coordinates is used with the origin centered at one camera



station and the Z axis passing through the second camera station extending vertically upwards. These are related to carriage coordinates through the desired scale factor; i.e., $x_o = X/m$ and $y_o = Y/m$ while $z_o = Z/m$ is used in the computer program. This coordinate system permits a common y_o value and x_o and z_o values that are simply displaced by the corresponding camera station separation components.

In this system of coordinates, the position of a given terrain point as it appears on the two photographs (assuming a distortionless lens) may be expressed by the relations

$$x = \frac{U_1X + U_2Y}{W_1X + W_2Y + Z} \tag{1a}$$

$$y = \frac{V_1X + V_2Y}{W_1X + W_2Y + Z} \tag{1b}$$

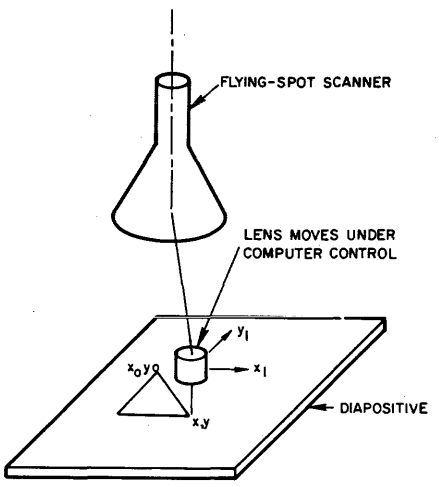
and, for the second photograph

$$x' = \frac{U_1'(X - B) + U_2'Y}{W_1'(X - B) + W_2'Y + (Z - \Delta Z)} \tag{1c}$$

$$y' = \frac{V_1'(X - B) + V_2'Y}{W_1'(X - B) + W_2'Y + (Z - \Delta Z)} \tag{1d}$$

Equations (1) are in normalized form; the origin of coordinates in the photographs is taken to be the nadir point, the point where the vertical through the center of the lens passes through the film (and hence where the image of the point directly beneath the camera appears). The origin of coordinates of the field of view, i.e., of the (X, Y, Z) system, is taken to be at the position of the lens for the first photograph; the displacement B of the X coordinate, and ΔZ of the height coordinate for the second photograph represent the change in position of the camera station for the two photographs.

Equations (1) permit the calculation of the coordinates on each of the photographs where a given spatial point (X, Y, Z) is to be found. In general, the positions (x, y) and (x', y') will not agree with the table position (x_o, y_o); the system permits the observation of the offset position through the use of a flying-spot scanner with an associated imaging lens whose position is under the control of the computer. The imaging system is shown in Figure 3. The flying-spot scanner is shown positioned over a



point (x_o, y_o) with the lens displaced by (x_1, y_1) so that a centered spot would be imaged at a desired position (x, y).

Four flying-spot scanners are used—one with each photograph and two for data printout. The latter are used with fixed lens so that the data is printed out at the position (x_o, y_o); i.e., at the properly scaled map coordinates of the area under observation at a given time. Data printout takes two forms: (1) An altitude chart exposed by computer control of the brightness of one scanner in accordance with the measured altitude; three brightness levels are used in a rotary sequence to show successive contour intervals; (2) A new photograph exposed by reproducing the image picked up by one of the photograph scanners and imaging it appropriately on the photosensitive film sheet.

During the setup operation, the photographs are mounted so that their axes agree closely with the machine axes. The computer then directs the system to move to the position of a point which can be identified easily and whose photographic coordinates are accurately known. The operator observes the scanned area as reproduced on a stereoviewer, basically a twin TV system having an electronically generated crosshair. Through Flexowriter control of the computer, the operator moves the lenses until the required point on the scanned area is centered on the crosshair; when this has been accomplished, the computer records the position of the point in system coordinates and then moves the scanned area to a second point where the operation is repeated. Once this has been accomplished, the computer modifies the coeffi-

cients of equations (1) (including a shift of the origin) and commands the system to move to a third "check" point as defined by spatial coordinates (X, Y, Z) so that the modified equations (1) are used. If the check point is well centered in the crosshairs, it is assumed that the correct relationships have been entered into the computer and the compilation process is started.

The compilation operation consists of a series of profiling runs in each of which x_o is constant and y_o proceeds in 0.010 in. steps. At each step, with x_o and y_o defined and an estimate of altitude available from previous measurements, the computer performs the arithmetic indicated by equations (1), subtracts x_o or y_o to obtain the required offset to locate the estimated point on the two photographs and outputs the resulting four values to corresponding digital-to-analog converters. The computer also outputs a signal corresponding to the altitude of the camera above the point under consideration and a signal to control the printing of the altitude chart. The analog system then takes over while the digital system proceeds to update the information for the next measurement point.

The rasters of the flying-spot scanners are centered on the designated areas by the servos acting in response to the corresponding d/a outputs. In addition, the rasters are individually controlled in shape and size so as to scan the photographs in one-to-one correspondence with the instantaneous position of the printout scanner exposing the new orthographic projection photograph. The process is best explained by the pertinent mathematics: let equations (1a) and (1b) be rewritten in the form

$$x = F(x_0, y_0, z_0) \qquad (2a)$$

$$y = G(x_0, y_0, z_0) \qquad (2b)$$

If the nominal orthophoto position (x_o, y_o) is varied by (dx_o, dy_o) the corresponding changes in x and y are given by

$$dx = \left(\frac{\partial F}{\partial x_0} + \frac{\partial F}{\partial z_0}\frac{\partial z_0}{\partial x_0}\right) dx_0 + \left(\frac{\partial F}{\partial y_0} + \frac{\partial F}{\partial z_0}\frac{\partial z_0}{\partial y_0}\right) dy_0 \qquad (3a)$$

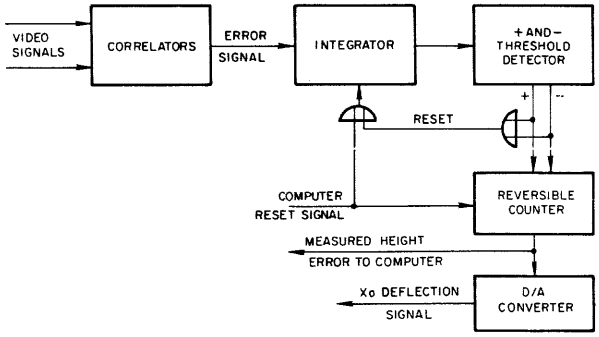
and

$$dy = \left(\frac{\partial G}{\partial x_0} + \frac{\partial G}{\partial z_0}\frac{\partial z_0}{\partial x_0}\right) dx_0 + \left(\frac{\partial G}{\partial y_0} + \frac{\partial G}{\partial z_0}\frac{\partial z_0}{\partial y_0}\right) dy_0 \qquad (3b)$$

In equations (3) dx_o is implemented as a fast (line) scan and dy_o as a slow (frame) scan. For the second photograph F and G are replaced by F' and G'. The current implementation includes only the scaling terms $\partial F/\partial x_o$ and $\partial G/\partial y_o$ as obtained by a low accuracy d/a from the computer. It has been demonstrated that an appropriate error signal for use in implementing the terrain slope terms $\partial z_o/\partial x_o$ and $\partial z_o/\partial y_o$ can be obtained by the analog system by comparing the altitude error signals on appropriate halves of the scan. It is anticipated that the scan equations will be more completely approximated in the near future with the remainder of the partials required to implement the scan equations (3) obtained from the computer.

With the two photographs examined at nearly corresponding areas and with the scanning proceeding along lines that are effectively in the direction of the camera separation, it is possible to use straightforward analog correlators to detect any height error as evidenced by a time delay of corresponding elements in the pair of video signals.

The height-error measuring unit is diagrammed in Figure 4. It consists of the correlation circuitry which yields the height-error signal, an integrator, a ± threshold detector, a reversible counter and an associated d/a converter providing an appropriate x deflection voltage to the photograph scanners. At the

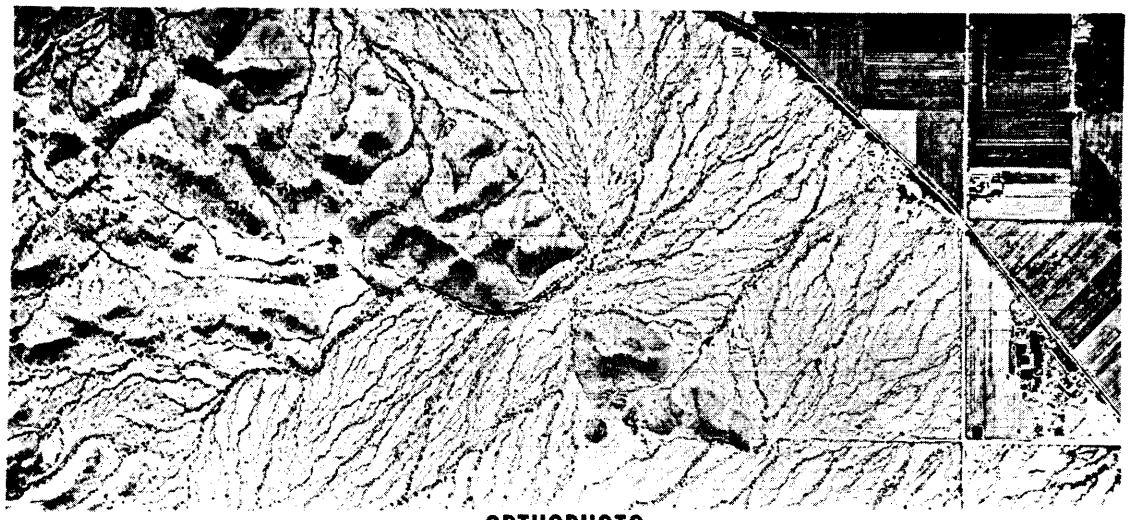


beginning of a measuring cycle, a pulse from the computer sets both the counter and the integrator to zero. As the scan progresses, if

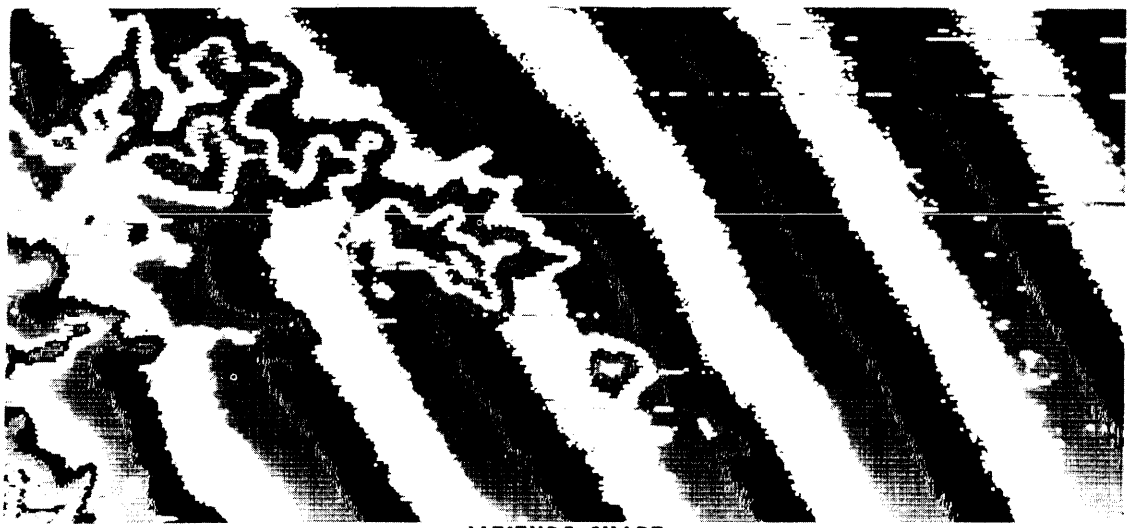
there is any error the integrator output will increase until the threshold is exceeded. This causes the reversible counter to step in the appropriate direction and the integrator to be reset to zero so that an independent error evaluation can again be made.

The count operates through the d/a converter to shift the photograph scan in a direction to compensate for the observed error. As the scan continues, any uncompensated error will cause further stepping of the counter until equilibrium is reached. At an appropriate point in the computational cycle, the measured height error is transferred into the computer and added to the original height estimate to correct the value in the computer memory. The cycle is then repeated for the next point in the profiling sequence.

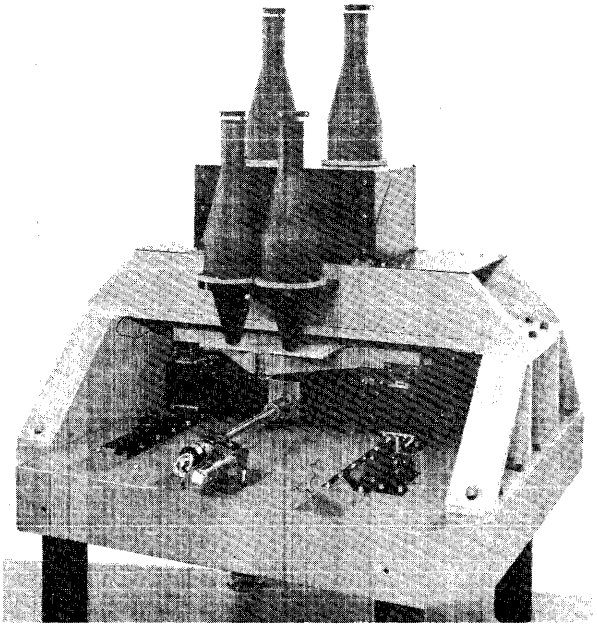
As the photographs are scanned the resulting video information is used to recreate the photographic element on the scanner used to expose the new photograph. In practice the original height estimate is sufficiently close to insure adequate accuracy in the resulting photograph. A height signal, in the form of one of a set of three brightness levels, is obtained from the computer and used to control a fourth scanner exposing the altitude chart. An orthophoto and corresponding altitude chart made by the equipment are shown in Figure 5. A photograph of the basic mechanical assembly for the system is shown in Figure 6.



ORTHOPHOTO



ALTITUDE CHART



The description of the system given above omitted many interesting details. Some of these will now be described.

COMPUTER PROGRAM

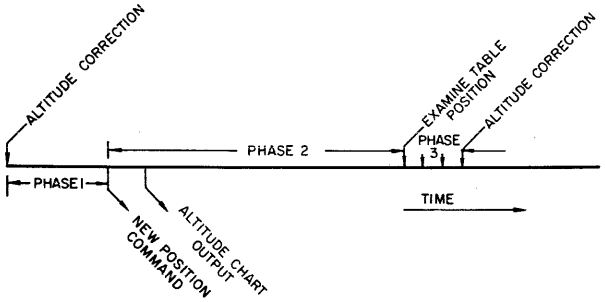
The timing problem for the computer program for a Y profiling operation is shown in Figure 7. It consists basically of three phases:

PHASE 1—Initiated by reading the altitude correction signal and followed by the computer preparing the new photograph position output command;

PHASE 2—The analog system operates to evaluate the height error while the computer prepares the data to expedite the Phase 1 operation;

PHASE 3—A waiting period for the analog system to move to a new position. The table position is read periodically during this period and the next cycle is initiated immediately on the signal indicating that a new position has been reached.

During the first phase the altitude error, measured at the last position, is added into the previously prepared denominators of equations (1), and these are divided into the previously prepared numerators. The quotients are then



added to a constant term and the result outputted through the digital-to-analog converters to the analog system. The analog system is idle during this period, so the time is made as short as possible.

The major part of the calculations are made in the second phase. The operation begins with the calculation of the output for the altitude chart. This is accomplished by assigning the current altitude to one of three levels in a rotary sequence. The result is outputted to the analog system where it is used to set the brightness of the altitude chart printout scanner. (Since the printout is made one cycle late, the analog printout is displaced one element ahead to compensate.) The numerators and denominators of equations (1) are then updated for the next cycle by adding a constant to each appropriate to the y_o increment being used (0.010 in.). The computer then calculates the position code for the next expected position (a two bit Gray code is used). This is then followed by one of a group of calculations that are rotated among six cycles, since they change very slowly compared to the accuracy requirement. Included here are corrections for distortion in the photography and the scale required by the analog system for scan adjustment.

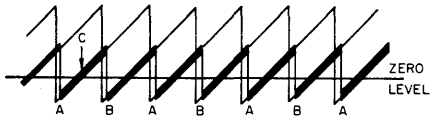
When a profile has been completed the carriage is moved over the desired amount in x_o and the numerators and denominators of equations (1) incremented a corresponding amount. The commands to increment the terms during the y profiling operation are then reversed in sign and the y motion started in the opposite direction.

STOP-MOTION SYSTEM

The speed requirement for the system is such as to make the design of a mechanical carriage that stops at each measurement point impractical, while the accuracy requirement demands

that measurements be made about well-defined positions in the photographs. These seemingly incompatible requirements are satisfied through the stop-motion element.

The stop-motion circuitry develops a pair of sawtooth voltages, as shown in Figure 8. These are triggered by alternate Gray code changes and have slopes that are dependent on the rate of motion of the carriage. If either of these, say A, is supplied to the flying-spot scanners, the result is to make the movement of the spot such that its image on the photograph appears stationary. The sawtooth voltage persists for two Gray code intervals—0.020 in.—and then shifts to permit a new area 0.020 in. behind to be examined.



In steady state operation, the output is switched between the two voltages A and B by computer command. Successful operation requires that each computer operating cycle take less time than the interval between Gray code changes. A given computer cycle might, for example, be completed at point C; the computer would then stall until the next Gray code change before providing the signal to switch to the alternate sawtooth. Thus, the two signals can be adjusted dynamically, independently of the computer, without danger that operation with the computer will change the calibration.

SERVO COMPENSATION

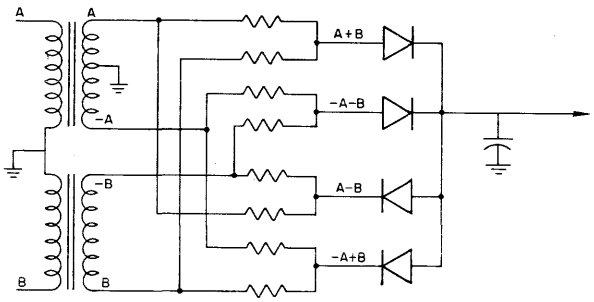
The speed requirements for the system also pose a problem for the rapid positioning of the scans to the required positions on the photographs. The displacement of the scans from orthophoto position is so large that it is not practical to use a completely electronic scan positioning system. As described earlier, and shown in Figure 3, primary positioning is accomplished using a pair of servo systems to position the lenses. The response of such servos operating by themselves is too sluggish to operate effectively in this application. For this reason the servo error signals are supplied to their respective scanners to compensate for any instantaneous error. Thus immediately after the computer outputs a new position command through the d/a, the system can start accumulating a meaningful error signal since the scan is centered on the desired areas.

The basic correlator used in the system is essentially a quarter-square multiplier. The configuration is shown in Figure 9. Operation is as follows:

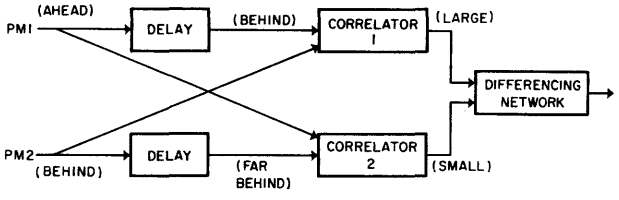
Assume the diodes shown operate as square law devices for signals of appropriate polarity. The two transformers are arranged to supply various combinations of the two input voltages A and B to the four diodes. If the output of a diode is expressible by $i = Ke^2$, where e is the input voltage and i the output current, then for the four diodes the sum is

$$i = K[(A + B)^2 + (-A - B)^2 - (A - B)^2 - (-A + B)^2] = 4KAB$$

so that the output is proportional to the instantaneous product of the two input signals. (Deviations of the diodes from square law does not significantly affect operation of the circuit as a correlator.) The integrated output, obtained from the capacitor, is then used as a measure of the correlation of the two input signals.



Two correlators are used in the height sensing circuitry along with a pair of delay lines, as shown in Figure 10. The figure illustrates the operation where the signal from the first photomultiplier is ahead of the signal from the second. For this condition Correlator 1 receives signals that are nearly coincident in time, and it therefore has a large output. Correlator 2, however, receives signals that are displaced in time so that it has a low output. The differencing network produces a corresponding output.



If the signals are coincident in time, the difference goes to zero, while if the direction of the differential reverses, the output reverses in polarity. The output is therefore appropriate to drive the integrator of the height-error measuring unit shown in Figure 4.

SLOPE COMPENSATION CIRCUITRY

Implementation of the photograph scanning signals described by equations (3) is dependent upon the derivation of suitable slope error signals (an early implementation used the computer to define the slopes, but the data for this is too noisy). The slope error signals have been derived from circuits analogous to the height-error sensor. The error in $\partial z_o/\partial x_o$ is determined by time-gating the photomultiplier signals so that for the first half of an x scan the signals are applied to the output with one sign and for the second half with reversed sign. If there is a uniform altitude error over the scan, it will be cancelled between the two halves; however, if there is a height-error differential over the scan (a slope error), the output will be appropriate to drive the slope error store to correct the slope in the system. Similarly, any error in $\partial z_o/\partial y_o$ is determined by time-gating the signal at the middle of the y scan.

CONCLUSION

The problem of automatically reducing stereo pairs of aerial photographs to contour intervals and orthophotos as a step in map making involves the manipulation of a large amount of data. The solution described retains the original photographs as the principal store for the process, thus avoiding the necessity of shifting data around in a digital store of hundreds of megabits.

Appropriate scanning equipment provides rapid access to the photographic store as needed, and straightforward analog techniques permit the processing of this information external to the computer so that the computer operation can be largely limited to those requiring accuracies not readily achievable in the analog system. Since the computer is in the system, it is expedient to use it for some low accuracy calculations that would otherwise unduly complicate the analog system—for example, the correction of distortion in the photography is made in the digital computer. It is believed that the system configuration represents a nearly optimum marriage of digital and analog techniques for the application.

A prototype of the Map Compilation System makes about fifty independent altitude measurements per second to an accuracy of better than one one-thousandth of the flying altitude. It has compiled a stereo pair in about an hour and a half, with very little assistance from the operator. A corresponding hand compilation would take much longer, and, in complex areas, would probably miss some information. The present system has adequately demonstrated the feasibility of automatic compilation. A second generation instrument is now contemplated that will be faster, more accurate, and more versatile. It will follow closely the operation of the present system.

ACKNOWLEDGEMENT

Achievement of a working system required a team effort. The contributions of George Miller, mechanical engineer, Glen Kimball in electronic design, and Jules Mersel in computer programming were particularly significant.

BIBLIOGRAPHY

1. R. D. ESTEN. "Automatic Contouring," *Photogrammetric Engineering* (March 1957).
2. R. E. WILLIAMS. "The Automatic Map Compilation System," *Photogrammetric Engineering* (March 1959).
3. G. L. HOBROUGH. "Automatic Stereo Plotting," *Photogrammetric Engineering* (December 1959).
4. W. C. CUDE. "Automatic and Semi-Automatic Mapping," *Photogrammetric Engineering* (April 1960).
5. U. V. HELAVA. "Analytical Plotter Using Incremental Computer," *Photogrammetric Engineering* (June 1960).
6. G. C. TEWINKEL. "Trends in Automatic Photogrammetry," *Photogrammetric Engineering* (September 1961).
7. E. C. JOHNSON. "Systems Design of a Digital Control Computer for an Analytical Stereoplotter," *Photogrammetric Engineering* (September 1961).
8. S. JACK FRIEDMAN. "AP-1—A New Concept in Stereoplotting," *Photogrammetric Engineering* (July 1962).
9. S. BERTRAM. "Automatic Map Compilation" *Photogrammetric Engineering* (January 1963).

AUTOMATIC READING MACHINE FOR TELEGRAPH SERVICE

W. D. Buckingham
Assistant Electronics Engineer
The Western Union Telegraph Company
Research and Engineering Department
Water Mill, L.I., New York

In the field of optical character reader research, emphasis heretofore has been placed on the development of high speed readers capable of handling large quantities of information at speeds sufficient to meet the input requirement of data processing and computer systems. Speed capabilities of presently available character readers range up to 25,000 words per minute.

Telegraph Service Requirements

Equipment of this type is not suitable for telegraph service, load concentrations at any one point in a network generally being insufficient to permit economic usage of reader capacity. Other economic factors affecting application are the necessity of providing a high ratio of spare reader capacity to protect continuity of service, and the need for special reperforator equipment to adapt the high speed output of the readers to a number of slow speed telegraph channels serving several points.

Western Union, recognizing the potential value to the telegraph industry of a low speed optical message reader, instituted a program of development several years ago. This project produced the new Western Union, Type 11343-A, Optical Character Reader shown in Figure 1. The reader automatically reads messages stored in a magazine at a rate of 16.2 characters per second, or 162 words per minute, converting them into five-unit code punched tape form. It is designed to provide the telegraph industry with a comparatively simple low-cost reader compatible with telegraph network speeds.

Loading

Referring to the photograph of the new machine shown in Figure 1, messages are placed in the slotted magazine, shown at the upper left hand side of the reading machine, one message

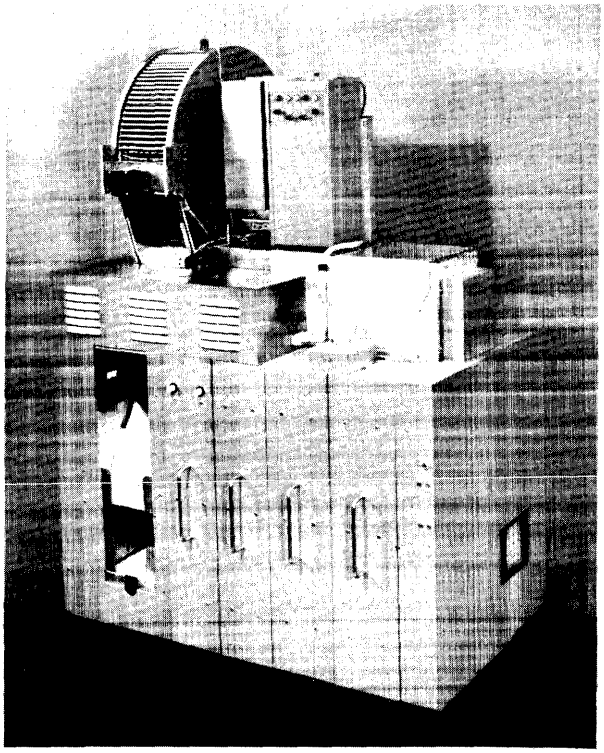


Figure 1. Western Union Optical Character Reader Type 11343-A.

per slot. They are automatically loaded, one at a time, in sequence, as the scanner calls for the next message. Twenty-four messages can be stored in the loader magazine.

When the scanner is empty, a message drops from the magazine and is automatically wrapped around the scanning drum. The drum appears at No. 1 in the schematic drawing of Figure 2.

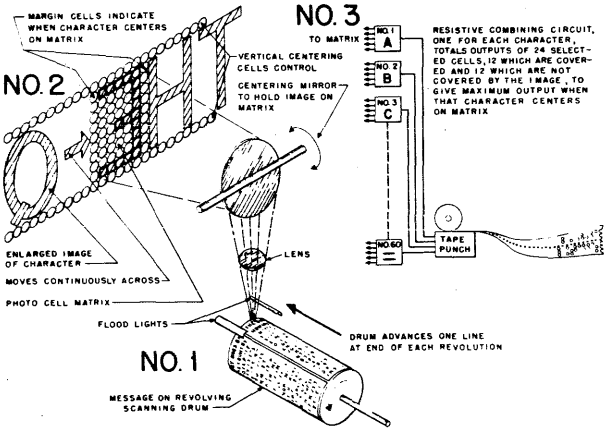


Figure 2. Pictorial Schematic of Reader.

Scanning

The scanning drum revolves at a uniform speed of 11.4 RPM as the copy is read. It is positioned to bring the typewritten characters, of the top line of the message, in sequence to the focus point of the projection lens. Since this section of the drum is brilliantly floodlighted by two 100-watt quartz-iodine lamps, enlarged images of the characters under the lens are projected onto the photocell matrix. They are magnified 25 times, or to a height of about 2½ inches.

The first revolution of the drum scans the top line of the message. The drum is then moved along its axis so as to scan the second line and then a new line on each subsequent revolution. If there is a short line, the machine scans the blank portion at five times normal speed.

A total of 170 low-cost cadmium selenide photoresistive-type photocells are used in the machine. Their speed of response is comparatively slow, but adequate for the 162 words per minute required of the machine. Their sensitivity, however, is very high, being of the order of amperes per lumen, the equivalent of a photo multiplier tube. Physically, they take the form of a cylinder ¼" in diameter and ½" long with the sensitive surface on the end. They are arranged with a compact rectangular 8 x 11 cell matrix of Reading Cells in the center of the field surrounded, on the top and bottom by long horizontal rows of Vertical Centering Cells, and, on each side by a vertical row of Margin Cells. This arrangement is shown at No. 2 of Figure 2.

The action of the drum, the lens and the centering mirror combine to produce an enlarged and reversed image of the character on the photocell matrix. During reading, the letter image moves across the matrix at a uniform speed. It is locked in the correct vertical position, balanced between the upper and lower rows of Vertical Centering Cells, through their control of the inclination of the centering mirror on its horizontal axis.

Letters on the copy, read by the machine, are spaced to produce a clear margin on either side of each character. The moment, when the moving image of a character may be covering only the Reading Cells, is indicated when both rows of Margin Cells are clear of any character image. Power is applied to the Reading Cells during this period only. At the instant when the image is centered on the Reading Cells, certain of the cells WILL be covered by the dark image of the character and their resistance will be high. Another group of cells WILL NOT be covered by the image and their resistance will be low. The division of the cells into these two groups of "Covered" and "Not Covered" cells is unique for each different character and is the basis of the recognition system used in the reader.

Recognition

A Reading Photocell circuit is shown in Figure 3. The cell is connected so that one terminal is fed positive battery through a resistor and the other terminal negative battery through a similar resistor. The junction of the cell and the resistor going to plus battery is termed the "Covered" connection point. Its potential to ground is strongly positive when the cell is covered by the black portion of the image and approaches zero when the cell is not covered by the image. The other cell terminal, called the "Not Covered" connection point, has a low

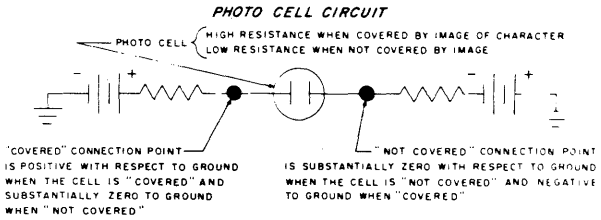


Figure 3. Reading Photocell Circuit.

negative potential to ground when the cell is not covered and a high negative potential when the cell is covered.

The reader has a printed-circuit "Resistive Combining" card for each of the different characters to be recognized. The cards are all alike and are shown in Figure 4. The output point on each is connected, through twenty four separate 2.2 megohm resistors, to the "Covered" connection point of each of a group of 12 cells selected from the "Covered" and to the "Not Covered" connection point of 12 cells selected from the "Not Covered" cells of the character to which the card is assigned. It has been found that twelve cells of each kind give sufficient information for positive recognition. If the cell selection is made properly, the output voltage of a given "Resistive Combining" card will be maximum positive when the image of its assigned character is centered on the reading matrix. Any other character produces an output which is substantially less on this card but a maximum on its own card. The voltage margin above the interference, varies from character to character, but averages more than a volt. The card, whose high output indicates that its character is the one on the matrix, fires its associated thyratron, which in turn causes the tape punch to punch the code for that particular character.

Auxiliary Functions

A space in the message being read is indicated and punched in the tape when the margin

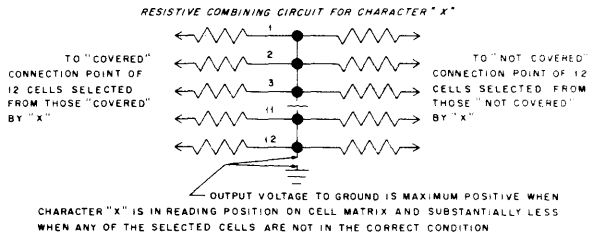


Figure 4. Resistive Combining Card Circuit.

cells on the entering side of the reading cell matrix fail to detect the passage of a character across them at the end of the proper time interval. Three or more spaces in sequence are recognized as the end of a line and the machine is shifted to high speed and advanced to read the next line.

A "letter" or "figure" shift is automatically inserted in the punched tape by the machine when needed. "Carriage returns" and "line feeds" are punched in the tape when symbols assigned to them appear in the copy being read.

An unreadable character is indicated when the entering margin cells detect a character and none of the "Resistive Combining" cards operate the punch. In this case, "Bust This" is automatically punched into the tape, the message unloaded from the drum, deposited in a special file and an alarm given to alert an operator who will process the messages manually.

A message that has been read completely is unloaded, time stamped and deposited in the "sent" message file.

The present machine is equipped with 60 "Resistive Combining" cards and can recognize the total of 60 capital letters, figures, punctuation and other symbols of one particular type font. One of the symbols acts to delete, or in effect erase, any character on which it is overtyped. Additional characters and fonts can be read by adding one "Resistive Combining" card for each new character.

Accuracy

The first Western Union Optical Character Reader is now undergoing accuracy tests. More units are being built so as to give the first test installation sufficient operating and spare reader capacity for economic evaluation. Accuracy tests, thus far, are favorable. It appears that the new low-speed, low-cost machine will achieve, in its speed range, accuracies quite comparable to those reported for the higher-speed, higher-priced units now available from other sources.

Application

The message reader is expected to have wide application in private wire network, as well as in our own services. In modified form it would be useful for automatically introducing a limited amount of printed information into data

processing or computer systems. Initially, copy to be read will be prepared using electric typewriters with one-time plastic ribbon and with Farrington self-check type so as to produce the highest quality copy. Lower quality copy may be used in the future if accuracy requirements can be met.

BIBLIOGRAPHY

A Reliable Character Sensing System for Documents Prepared on Conventional Business Devices. David H. Shepard, Pickard F. Bargh & Clyde C. Heasly, Jr. IRE Wescon Convention Record, 1, part 4 (1957) 111-120.

Linear Decision Functions with Application to Pattern Recognition. W. H. Highleyman. Proc. IRE, June 1962, p. 1501.

Image Processing with Optical Panels. H. O. Hook & H. Wernstein, Electronics, December 12, 1962, p. 35.

Print Reader Recognizes Variety of Fonts. G. L. Shelton, Ibid., p. 58.

The Simulation of Cognitive Processes. R. L. Simmons and R. F. Simmons. IRE Transactions on Electronic Computers, September 1961, p. 462. A bibliography of 460 references on the subject of character recognition.

Simulation of Three Machines which Read Rows of Handwritten Arabic Numbers. L. A. Jameulsky. Ibid., p. 489.

An Analog Method for Character Recognition. W. H. Highleyman. Ibid., p. 402.

Computer Synthesis of Character Recognition Systems. D. N. Freeman. Ibid., December 1961, p. 735.

A RESEARCH LABORATORY FOR PROCESSING AND DISPLAYING SATELLITE DATA IN REAL TIME

R. H. Spitler and B. K. Kersey
Lockheed Missiles and Space Company
Sunnyvale, California

Section 1

BACKGROUND

Sometime ago, it became apparent to the Management of Lockheed Missiles and Space Company (LMSC) that a real-time data-processing system would be required for one of the specific satellite weapon systems then being developed at LMSC. Following discussions with personnel of the Air Force Space-Systems Division, Los Angeles, California, it became obvious that many technical questions would have to be answered before approval for the construction of a world-wide command and control system would be granted.

Systems being developed throughout the country to process satellite data in real time had one thing in common: they were—and still are—expensive to build, operate, and maintain. In many cases such systems were installed before adequate preliminary studies could be made. With this factor in mind, we suggested that perhaps the easiest and quickest way to provide answers was to build a small Research Laboratory able to simulate accurately various portions of a world-wide data-collection and processing system.

Immediate implementation was one of the key requirements, so with Air Force approval, construction of the Laboratory was started on 12 March 1962 when the first wall was erected in the LMSC Sunnyvale facility. By 30 June 1962—only 90 working days after go-ahead—the Laboratory was in operation. Much of the credit is due companies such as IBM, Ampex, and General Dynamics, who worked with LMSC during this comparatively short time span to design, build, and deliver the input, interface, computer, and display equipments.

LMSC was aided in no small part by the experience it had gained during extensive real-time processing of data from operational and experimental satellites. In particular, the LMSC-designed and -constructed equipment was based in some cases on actual equipment presently in operation throughout the world, and in other cases on advanced but tested designs. In other words, LMSC stayed within the state-of-the-art. Fortunately, in the area of programming personnel, we were able to obtain highly qualified specialists.

While the primary objective of the Laboratory is to realize substantial savings by proving designs on a small scale prior to implementation, anyone who operates a computer such as the IBM 7090 knows the expense of such a machine when idle. Therefore, an operational philosophy was devised to reduce the cost per job processed on the computer. To implement this approach the Laboratory operates in two modes: first as a laboratory to simulate by experiments the real-time data-processing systems under study. This mode has top priority. The second mode prevails when experiments are not being conducted. Then "fill jobs" are obtained from the centralized computing area

and handled in a standard batch-processing mode. As a result of this practice, the cost to the Air Force for this working facility has been reduced well below original estimates.

Section 2

LABORATORY SYSTEM

The Laboratory was designed to simulate the operation of both a Tracking Control Center and a Remote Readout Station. It was intended that the following benefits would be realized through the use of this simulation facility:

- New techniques for reliable assessment of data processing in near real time
- Computer and display equipment specifications for the Tracking and Control Center
- Computer programs suitable for processing satellite data in real time
- Development of improved man/machine relationships and operational procedures
- Improved detection capability under simulated operational conditions
- Detailed requirements for the Satellite Communication System

2.1 *System Description.*

To make the Laboratory operational as soon as possible and to enable the programming personnel to concentrate early on the basic programming tasks of simulation, an IBM 7090 Data Processing System was selected. The IBM 7090 Computer serves as the central processing element to control and process the raw satellite data into forms suitable for dynamic presentation on display equipment in near real time.

The IBM 7090 is used to establish data processing routines for the Tracking and Control Center and Remote Station operation. Basically, it is similar to the standard 7090 Batch Processor. Additionally, in this case it has an IBM 7281-II Data Communication Channel with a direct data device connection. Attached to this channel are various subchannels, engineered by IBM to take a variety of inputs, such as the timing equipment and data compressor outputs, and convert them to a format suitable for entry into the IBM 7090. Five outputs from the system are available for inspection:

- General Dynamics displays used to develop the initial programming concept and provide devices for demonstration purposes
- IBM displays allowing operators to view the output of one satellite and determine validity of computer decisions regarding recognition of patterns
- IBM displays for operators to determine validity of computer decisions regarding pattern correlation and identification, input data to this device being that which survived the pattern recognition function
- Modified PAM Analog display which shows the raw data collected by the satellite
- Off-line listings on IBM 1401 Computer

Figure 2-1 indicates the interconnection of the major components. Major components are the PAM Data Conversion Station, Data Compression Unit, IBM 7090 Computer, and the displays. Inputs to the system are:

- Frequency-modulated signals from the FR700 Tape Machine, demodulated and digitized before entry into the IBM 7090
- Parallel digital data from the modified Ampex FR100 Tape Machine
- Serial digital data (simulated telephone input at 1200 bits per second) from the Communication Simulator
- Parallel digital data from the IBM 729 Tape Machine

2.2 *Operation.*

The Laboratory operates under two basic modes of operation: the Laboratory Mode and the Batch Processing Mode. In the Laboratory Mode, one or more of the external devices is connected on-line with the computer to process one or more simulation tasks. The Batch Processing Mode approximates a commercial data-processing and computation center.

Laboratory Mode. Within the Laboratory Mode it is possible to exercise any combination of the equipment described in Section 3. Table 2-1 shows 14 possible combinations to illustrate the more important configurations used for experimental purposes.

In order to provide coordination between the operation of the digital computer and external devices, it is necessary to provide separate operating procedures to ensure that each piece of

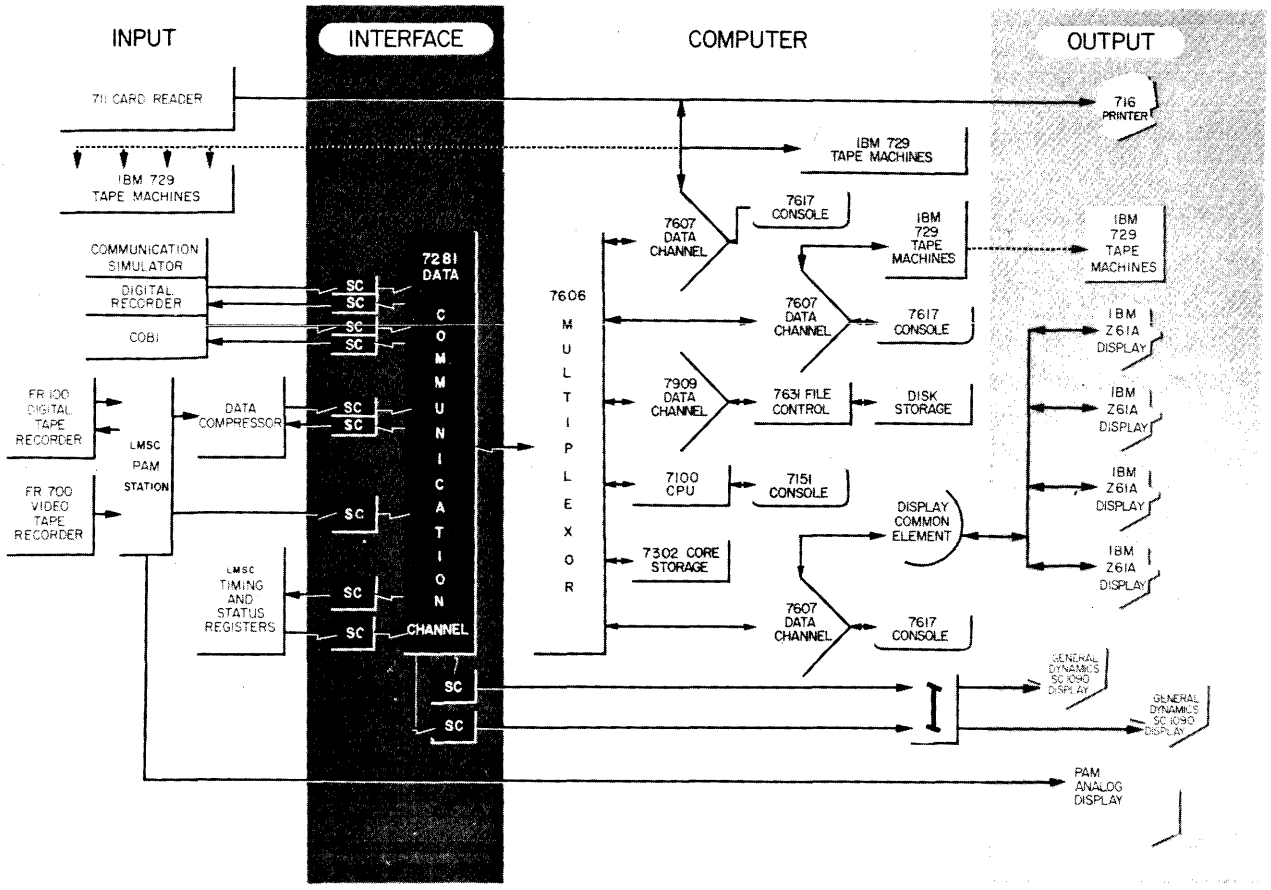


Figure 2–1. Flow Diagram of Data-Processing Laboratory.

equipment is operational at the scheduled time of the experiment. To facilitate the integration of these two parallel processing capabilities, a Test Director Console manned by the Test Director was established. The Test Director proceeds through a countdown cycle prior to beginning the experiment. This procedure ensures that all equipment is operational and minimizes delay times that could be caused by malfunction or setup delay of the various devices.

1.	FR 700	PAM	IBM 7090	G. D. DISPLAY
2.	FR 700	PAM	IBM 7090	IBM DISPLAY
3.	FR 100	PAM	IBM 7090	G. D. DISPLAY
4.	FR 100	PAM	IBM 7090	IBM DISPLAY
5.	FR 700	PAM	DATA COMPRESSOR	IBM 7090	G. D. DISPLAY
6.	FR 700	PAM	DATA COMPRESSOR	IBM 7090	IBM DISPLAY
7.	FR 100	PAM	DATA COMPRESSOR	IBM 7090	G. D. DISPLAY
8.	FR 100	PAM	DATA COMPRESSOR	IBM 7090	IBM DISPLAY
9.	IBM 729			IBM 7090	G. D. DISPLAY
10.	IBM 729			IBM 7090	IBM DISPLAY
11.	TRIPLE ROS* SIMULATOR			IBM 7090	G. D. DISPLAY
12.	TRIPLE ROS SIMULATOR			IBM 7090	IBM DISPLAY
13.	SINGLE ROS SIMULATOR			IBM 7090	G. D. DISPLAY
14.	SINGLE ROS SIMULATOR			IBM 7090	IBM DISPLAY

*ROS-READOUT STATION.

Table 2-1
OPERATING COMBINATIONS

Intercommunication between the operators of the various equipments and the Test Director is provided by means of a Public Address System and an Intercommunication System. The Public Address System is used to alert operators to man their stations or to call in maintenance engineers if equipment malfunctions develop. The Intercommunication Equipment is used to provide direct communications during the experiment. As part of the experiment, primary diagnostics (described in Section 4) are executed before and after the run. This ensures that the equipment is functioning properly and validates the results.

Batch Processing Mode. While the primary objective of the Laboratory is to provide operational support of project experiments, there is a basic requirement to operate the facility efficiently and provide development facilities for

programming personnel. For these reasons the Laboratory is often operated in the Batch Processing Mode.

Section 3

EQUIPMENT AND FUNCTION

3.1 *Data Acquisition.*

This is achieved by means of a PAM General-Purpose Ground Station, Data Compressor, Communications Simulator, and Timing System. The equipment is used as a data source for the various signals required to simulate the components of a command and control system.

PAM General-Purpose Ground Station. This is a versatile, general-purpose unit capable of processing a variety of pulse-amplitude-modulated telemetry signals. Within the Laboratory it is used as the primary source of simulated data for use by the other laboratory equipment. Its major components are an FR700 Wideband Magnetic Tape Recorder, a Pulse-Amplitude Modulated Signal Detector and Subcarrier Discriminator, Synchronization Signal Separator and Digitizer, and an FR100 Digital Magnetic Tape Recorder.

Because it can handle many data channels per frame, sampled at high rates, this station (of which 10 are currently in use) is singularly suited to the laboratory. The pulse train presented is synchronized, separated into individual data channels, and formatted into digital eight-bit-per-channel parallel outputs.

The FR700 Wideband Tape Unit records and plays back the PAM-FM raw data for use by the Laboratory System. The PAM Signal Detector serves to remove the carrier wave from the data recorded on the magnetic tape. The detector then presents an analog signal, modulated by the PAM pulse train, to the Synchronization Signal Separator and Digitizer. This unit separates the synchronization pulses from the applied PAM pulse train, normalizes the data by utilizing calibration channels in the pulse train, and finally digitizes the individual data channels for either recording and/or further processing by the Data Compressor.

Data Compressor. This unit is capable of accepting PAM telemetry data from the PAM General-Purpose Ground Station. The high data rate is reduced and compressed to a rate compatible with a transmission system by thresholding all invalid information. The qualified data is entered into the 7090 Computer through one of the IBM 7281-II data subchannels. Thresholding data is entered into the Data Compressor from the 7090 through another IBM 7281-II data subchannel.

Two modes of operation are available. The first is a Status Operational Mode whereby only changes in data status, that is, in-limits to out-of-limits or vice versa are sent to the Computer. The second mode sends all out-of-limits conditions to the Computer. The modes are selectable for each data point in the telemetry wave train. Both normal telemetry data, such as azimuth information and payload data signals, are processed. Also, upper and/or lower limit comparisons are possible.

Communications Simulator. This equipment accepts simulation tapes recorded on IBM 1401 and 7090 Computer Systems. The information recorded on these tapes simulates remote readout station data transmissions. It can simulate actual data transmission using a COBI Modem driving a telephone line, incorporating noise and other data representing the real-life situation. Or it can simulate multiple communications lines transmitted in parallel to the IBM 7090 Computer simultaneously for processing.

The specific tape drive associated with the Communications Simulator is capable of accepting simulation tapes that have been recorded on the IBM 1401 or 7090 Computer Systems. Such tapes contain information representing six data transmission systems from remote readout stations. All six channels may be fed directly to the 7090 Computer System through a subchannel of the IBM 7281-II Data Communication Channel to simulate the ground network operation. Or one channel may be sent to the 7090 Computer through the Communication Simulator Equipment (COBI) for evaluation and development of the data transmission system. The Communications Simulator is designed to be operated manually or under computer control.

Timing System. The Timing System has two operating modes. One mode is used for recording simulated data and timing signals, and the other mode decodes recorded timing signals. In the recording mode, parallel data and timing signals are accepted from the IBM

7090 Computer via the IBM 7281-II Data Communications Channel. The data and synchronizing signals are recorded on 11 digital tracks of the FR100 Tape Recorder. The timing signal is serial-encoded, placed on a modulated carrier and recorded on a separate analog track. In the reproduce or playback mode, serial analog timing signals from the PAM General-Purpose Ground Station recorders are demodulated and decoded into a parallel digital format for use in the computer.

During both modes of operation, registers are available for recording time and for providing time-controlled computer interrupt signals. Time register outputs are available to the computer for processing and display by the remote timing displays. An accurate time base is provided for timing various functions throughout the laboratory.

The Timing System also includes a set of Status Monitor Registers and a Control Register. The Status Registers monitor the operational status of various equipment throughout the laboratory and present this information to the IBM 7090 Computer on demand. The Computer can also control the operation of certain equipment through the Control Register.

3.2 *Interface*

7281-II Data Communication Channel. The IBM 7281-II Data Communication Channel expands the input/output capabilities of IBM 7090 Data Processing Systems. This channel provides direct connection to a variety of input/output devices operating at various speeds for real-time applications. Like the IBM 7607 Data Channel, the 7281-II transfers data to and from the core storage through the Multiplexor in a 36-bit parallel fashion, concurrent with operations within the central processing unit. Data transmitted between core storage and real-time equipment passes through the 7281-II Channel. Operation is initiated by the execution of a single new instruction in the 7100 Central Processing Unit.

Once started, the 7281-II channel operates independently of the main program being executed in the central processing unit. The 7281-II channel controls the quantity and destination of all data transmitted between storage and real-time equipment. Data transmission to and from storage is in 36-bit (or less) parallel fashion. Within the IBM 7281-II Data Communication Channel there are 11 subchannels. Each subchannel has been uniquely designed to provide an interface between the 7090 Computer and the external device and to perform an input or an output interface. For the Data Compressor, two subchannels are used. The input subchannel transfers data to the 7281-II at a high-burst data rate which is stored in a block of core storage cells within the Computer. If the volume of data exceeds the limit of one block, the IBM 7090 is interrupted and the data is fed to an alternate block. This allows the programmer to process the first block of data during the time that the second block is being filled. The output subchannel for the Data Compressor allows the Computer to send limit data to be used by the Data Compressor for thresholding inputs. It is also used to send command functions for execution by the Data Compressor.

Two identical output subchannels are available to drive the SC 1090 Displays. Data is fed from the 7090 at the demand rate of display, allowing the display equipment to present pictorial information at a flicker-free rate.

One input subchannel is available to handle the direct data from the PAM General-Purpose Ground Station. This subchannel packs four data samples into a computer word before transmitting the data to the 7090, thus allowing more processing time between data inputs.

One input and one output subchannel are available to interface the Timing Equipment. The input subchannel to the Timing Equipment uses a single word buffer and interrupts the computer each time a word is submitted to the 7090 Computer. The output subchannel can work in two (2) modes: The first mode submits data words to set the time registers, the second is used to provide data for recording simulated data on the FR100.

There are four subchannels available to interface the Communications Simulator equipment. One input subchannel is used to provide an interface with the COBI Modem or with a single tape channel input. The other input subchannel is used to provide up to six channels of data simultaneously. One output subchannel is used to transmit data to the COBI Modem, while another subchannel is used to command the Potter Tape Drive machine within the Communications Simulator.

The demand transfer rate for all of the above subchannels, except the Communications Simulator interface, exceeds the transfer rate of 20,000 computer words per second.

3.3 *Computation*

General Computer. The Type-7100 Central Processing Unit contains the arithmetic and stored-program control circuits for the 7090 system. The Type-7151 Console Control Unit is a separate unit which provides centralized control of the 7090 Computer System. It contains indicator switches, key and register displays. Channel indicators are provided, and the register displays have been grouped for operator's convenience.

The Type-7302 Core Storage has a capacity of 32,768 thirty-six-bit words with random access time for extraction or storing of each word not exceeding 2.18 microseconds. The Type-7606 Multiplexor provides the only access to core storage. It accomplishes the necessary data and address switching to transmit data to and from core storage. The data channels and the central processing unit transmit all data to and from core through the Multiplexor.

Input/Output Equipment For Computer. The following components comprise the input/output connection to the computer from the real-time and supporting equipment. The equipment is connected on-line to assist in performing real-time simulation.

The Type-711 Card Reader is used to enter instructions and data into the 7090 Computer. It is controlled by the computer and reads cards at the rate of 250 cards per minute.

The Type-729 Model VI Magnetic Tape Drives are each capable of storing or reading almost 4 million computer words of information on each reel of tape at a rate of 15 thousand 36-bit words per second. Tape drives are used to store status and historical data, executive programs, support routines, test programs, test program data and temporary storage. Additional tape drives are used for assembly and checkout of programs and the evaluation of simulation runs. They provide the following:

- Programming systems and library storage
- Scratch areas for intermediate storage
- Coupling and buffering information between other components of the system
- Extension of memory for large programs or data storage
- Checkpoint and restart
- Problem program data

The IBM 729 Model-VI Tape Drives provide the following:

- Data Storage
- Storage of application programs
- Working storage
- Simulation output
- System protection

This enables the central processing unit to accomplish its functional assignments on a more timely basis. These uses are applicable to each of the various types of simulation.

The Type-716 Printer is a line printer for recording information from the computer in printed form at 150 lines per minute. The printer is used to provide on-line messages during simulator runs and also to notify operators when to control and change parameter characteristics for a given run.

The Type-7607 and 7617 Data Channel and Console channels are connected to the 7090. When information is to be processed on magnetic tape, one channel is used as an input channel and the other as an output channel. The Type-7607-III accommodates the input functions. It includes the 711 card reader and the 716 printer in addition to eight 729-VI Tape Drives. The Type-7607-IV Data Channel accommodates the output functions and is connected to the eight output tape drives. The Data Channel enables the 7090 to initiate and monitor input-output operations, but the Computer need not be involved with the detail data handling steps between the input/output devices and the computer memory. The Data Channel provides for the transmission of data between the Input/Output Units and Core Storage.

The Type-7617 Data Channel Console provides for greater input/output flexibility, and efficiency between the 7090 and its operator. One Data Channel Console is used with each Data Channel.

Mass Memory. The memory equipment provides real-time storage for data to be displayed or processed and receives data from the IBM 7090 Central Processing Unit. The information is then made available to the Display Equipment or Central Processing Unit by answering

requests for specific information. The equipment consists of a Disk Storage, File Control Unit, and a Data Channel.

The Type-1301 Disk Storage File provides storage for 56 million 6-bit characters with a transfer rate of 90,000 characters per second. It is used for storing Display Data, Programming Systems and Library Routines. It also provides working storage such as scratch files and tables, core memory extension for data requiring more rapid access than possible from tape units and less rapid access than required from the core memory, and storage of diagnostic routines.

The Type-7631 File Control Unit provides the interface circuitry between the 7909 Data Channel and the 1301 Disk File for access to the Disk File memory addresses.

The Type-7909 Data Channel provides the interface between the disk memory and other units of the computer and display equipment.

Peripheral. In order to provide a complete, self-contained configuration within the laboratory, the IBM 1401 Data Processing System was selected to provide peripheral support. The 1401 system provides the following support services:

- Card-to-Magnetic-tape transfer of programs and data
- Tape-to-Printer write-out of test results
- Tape-to-Card transcribing of programs for reassembly or permanent storage

The system comprises a Type-1401 Processing Unit which contains core storage and performs all the machine logic. A Type-1402 Card Read Punch has two card feeds that enable simultaneous punched card input/output operations. It has a rated reading speed of 800 cards per minute and a punch speed of 250 cards per minute. A Type-1403 Printer has a rated speed of 600 lines per minute with 132 positions per line and 48 different characters per position. Type-729-V Magnetic Tape Drives are also provided.

3.4 *Display*

Direct-View Data-Display—General Dynamics SC 1090. Two direct-view data-display consoles are in use in the laboratory. These displays are general-purpose units designed to provide visual outputs from digital data systems. The design is based upon the Charactron shaped-beam tube, which presents precisely shaped alphanumeric and special symbols directly on its 19-inch diameter tube screen and can be viewed in ambient light.

When equipped with all options, this display can present symbols, vectors, and up to four-line formats of four characters each plus a vector, or combinations of all three. Up to 21,000 random symbols per second can be presented. Operator-controlled category selection allows the inhibition of unwanted data to prevent clutter, and feature selection allows inhibition within a category. Expansion and off-centering permit viewing any portion of the screen in expanded mode with a resolution of 2,000 lines. Test patterns can also be called up for diagnostic purposes.

Display System—IBM Z61A. The principal features of the Display System are:

- Fast response time
- Manual inputs by use of a light pencil and push buttons
- Internal automatic display generation
- Expansion capability of 1, 2, 4, 8, 16, and 32 times and bright display in ambient light
- Off-centering to virtually any point where data is displayed
- Solid-state circuitry except for Charactron display tube
- Flexibility in operator selection of data

The unit contains a 2,048-word, 24-bit-word core memory. The console accepts data at a rate of one message each 396 microseconds with the capability of accepting no more than 152 messages in 2.5 seconds and can display one message every 396 microseconds. Each message is displayed at a repetition rate of 5 to 100 cycles per second, depending on the amount of information accepted by the console.

Fixed geography messages are accepted at a rate of one each 396 microseconds with a maximum capability of accepting 72 messages. The fixed geography messages are displayed each 396 microseconds and are displayed only once per display cycle (2.5 seconds).

Digital display data is accepted at a rate of one message each 66 microseconds with a maximum of 96 messages provided for in the memory. Display data is shown at a rate of one message each 66 microseconds.

PAM Analog Display. The PAM Analog Display was originally constructed by Baird Atomic Corporation and has been modified by Lockheed Missiles and Space Company to perform a specific Laboratory function. The original equipment consisted of two cabinets containing an externally-viewed 19-inch PPI Display Tube with an additional PPI Tube mounted internally for use with a closed circuit TV system. In addition, Oscilloscope, Controls, TV Display, and Command and Status Equipment were also part of this unit. Modifications for Laboratory use consisted of removing the Closed Circuit TV system, the Command System, and the Status Display System.

Functionally, this display unit accepts the pulse-amplitude-modulation signal from the PAM Ground Station accompanied by the various frame, block, and subframe synchronized signals. These signals are displayed as vehicle-stabilized PPI presentation and as either amplitude versus elevation or as amplitude versus azimuth on an oscilloscope. A photo camera is available to record successive sweeps as they are presented on the internal PPI Display.

Section 4

DIAGNOSTIC PROGRAM

The purpose of the diagnostic program is to accomplish the following:

- Determine that the total system is working satisfactorily before conducting experiments
- Upon determining that the total system is not operating, to find which subsystem has failed
- Upon determining the failure of the particular subsystem, to locate the faulty unit within that subsystem
- Determine that all equipment is working satisfactorily during running of experiments

The programs which will fulfill these purposes have been classified as Static Diagnostics and Dynamic Diagnostics.

4.1 *Static Diagnostics*

A Static Diagnostic is a program which allows operation personnel to determine the failure of the entire system or a specific subsystem prior to start of an experiment. This particular phase of the task has been divided into primary, secondary and tertiary diagnostics.

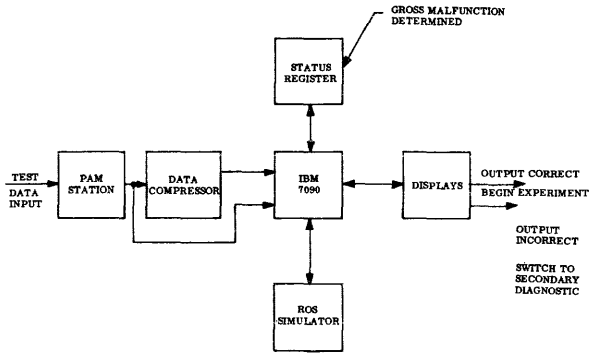


Figure 4–1. Primary Static Diagnosis.

Referring to Fig. 4-1, it can be seen that, for the primary case, *known* test data is fed into the PAM Station and checked in the IBM 7090 for validity. Simultaneously, the computer generates display data to check the validity of the display equipment. If the PAM and display data paths are correct, then the experiment proceeds. If the output is not correct, then the equipment switches automatically to the Secondary Diagnostic.

During the Secondary Diagnostic Check, as shown in Fig. 4-2, each major subsystem is tested to determine whether or not it has failed, by inserting test data which exercises the equipment through wider limits than is required in the primary case. To provide a rapid means of checking this equipment complex, a device called a Status Register has been installed.

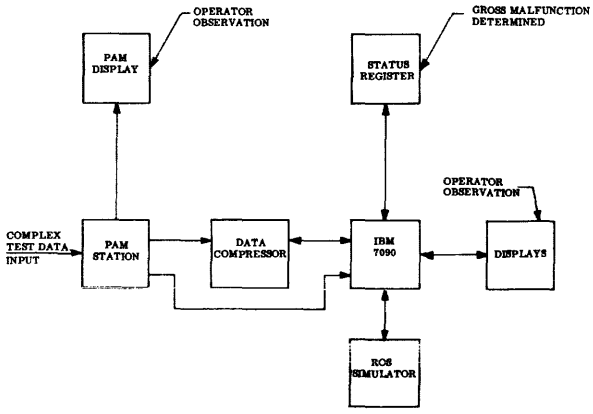


Figure 4–2. Secondary Static Diagnosis.

The function of the Status Register is to collect signals indicating that specific items of equipment are operating satisfactorily and to report when gross malfunctions occur. For example the following is stored in the register:

- Status of the synchronization quality in the PAM Station
- Density, file protect, and ready signal from the IBM 729 Tape Machine
- The reproduce signal from the FR700 Tape Machine

Thus, the Status Register is a series of flip flops that indicate the correct condition of each connected piece of equipment. Several Status Registers are available and physically housed in the Timing Equipment. As an economic consideration, the same circuitry for accessing Timing data to be fed to the computer via the IBM 7281-II is also used to access the Status Register. Additionally, during the exercise of equipment throughout its limits, each operator observes the various displays and measuring instruments to assist in ascertaining the particular subsystem which has failed.

Upon location of the particular subsystem which has malfunctioned, the Tertiary Diagnostic is started. The Tertiary Diagnostic for the PAM Station, Data Compressor, and Display devices, as shown in Fig. 4-3, merely requires that the Laboratory maintenance people be notified. If the failure appears to be in the IBM 7090 computer, then IBM Customer Engineers are notified and they begin detailed internal diagnostics of the computer.

4.2 *Dynamic Diagnostics*

Dynamic Diagnostics may be defined as the ability to determine if the equipment is operating satisfactorily during an actual experiment. The equipment can be exercised during an experiment by using either of two approaches.

When known input data is fed into the system (Fig. 4-4), it is possible to validate the equipment from "a priori" knowledge of the data input. Many experiments are available to give known outputs on the various displays. After a particular experiment, such as a raid or a cloud cover pattern, has been checked out, it is possible to rely on the display operators to ascertain whether or not the equipment is working properly. However, when nonpredictable data, such as actual target data is fed into the system, this "a priori" knowledge is no longer available.

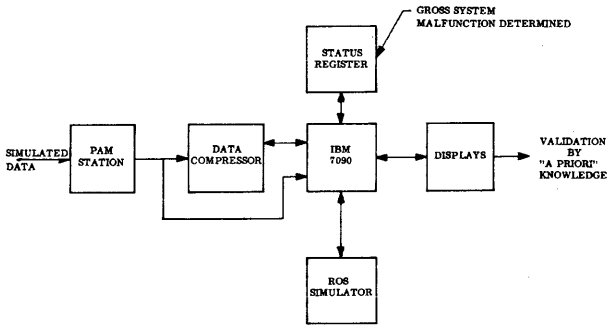


Figure 4-4. Dynamic Diagnosis Utilizing Known Input Data.

During this condition, as shown in Fig. 4-5, the Status Register becomes a rather important tool because it allows the determination of any gross malfunction in the system while an experiment is being run. At present, the Status Register holds four 36-bit computer words. An

Problem area	*Solution*
1. PAM Station 2. Data Compressor 3. ROS* Simulator 4. Situation Display	Call Laboratory Maintenance personnel, who will begin detailed component check of specific subsystem.
5. IBM Display 6. IBM 7090 Computer 7. IBM 1401 Computer	Call IBM Customer Engineers, who will begin detailed internal computer diagnostics.

* ROS—Readout Station

Figure 4-3. Tertiary Static Diagnosis

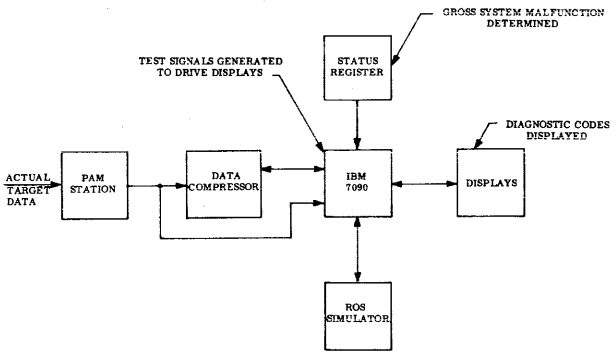


Figure 4–5. Dynamic Diagnosis With Non-Predictable Data Input.

error anywhere in the first computer word causes the experiment to stop by making it indicate a catastrophic-type failure such as the reproduce switch on the FR700 failing to actuate. Word "two" stops the experiment if two errors are indicated. Word "three" stops the experiment if five errors are indicated, while errors in word "four" do not stop the experiment, but a complete error record is maintained.

Utilizing this type of decision element, it is relatively simple to isolate the particular part of the PAM Station, Data Compressor, or Display that is inoperative. Since it is also possible to monitor various portions of the IBM 7090 Computer, such as the settings on the IBM-729 Tape Machines, catastrophic failures in the computer can be readily determined. In addition, various programs are available which cause test signals to be inserted by the computer into the various display devices. As long as this test data, which is inserted between processing times, is observed by the operator in specific portions of the display, the experiment is allowed to continue. Variations of this method cause different diagnostic codes to be displayed which give the operator some insight into the reliability of the observed data.

Section 5

CONCLUSION

The research facility as described is now operational within the Research and Engineering Branch of the Lockheed Missiles and Space Company (LMSC). Preliminary results indicate that the philosophies outlined herein are valid. Since the facility was first activated, equipment designs have been successfully tested, and the techniques used in developing the computer programs have been refined.

LMSC's experience in this area has shown that, whenever possible, a modest facility such as this should be constructed before a large scale, world-wide system is undertaken. In addition to reductions in time, manpower, and equipment, the financial gains are more than substantial.

A REAL TIME MULTI-COMPUTER SYSTEM for LUNAR AND PLANETARY SPACE FLIGHT DATA PROCESSING

W. R. Hoover, A. Arcand and T. B. Miller
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

I. INTRODUCTION

The California Institute of Technology, Jet Propulsion Laboratory (JPL) is responsible for National Aeronautics and Space Administration projects for the unmanned scientific exploration of the moon, the planets, and interplanetary space. The use of unmanned spacecraft has resulted in a need for rapid processing, display and analysis of data from the spacecraft and from the tracking equipment in order that ground control, using transmitted commands to the spacecraft, may be accomplished. Commands to the spacecraft are for (1) instructions for midcourse and terminal maneuvers, (2) instructions for telemetry and mode control, and (3) instructions for deployment and activation of spacecraft subsystems.

The general data acquisition, processing, and display problem for space flight support is represented in Figure 1. The data handling problem is logically divided into four phases:

1. Spacecraft to Deep Space Instrumentation Facility (DSIF) tracking sites,
2. DSIF on-site processing,
3. DSIF to JPL communications, and
4. Central acquisition, processing and display.

The central processing facility provides information to the engineering, flight path and scientific analysis groups and to flight operations director. A centralized command and control system approach has been established for the execution of space flight operations at JPL. An integrated facility is now under construction at JPL, called Space Flight Operations Facility (SFOF), and a multi-computer system providing on-line acquisition processing, display and control is being developed for data processing support.

The criteria which governed the design of this system were:

1. Provide automatic on-line acquisition and conversion of all data for automatic processing.

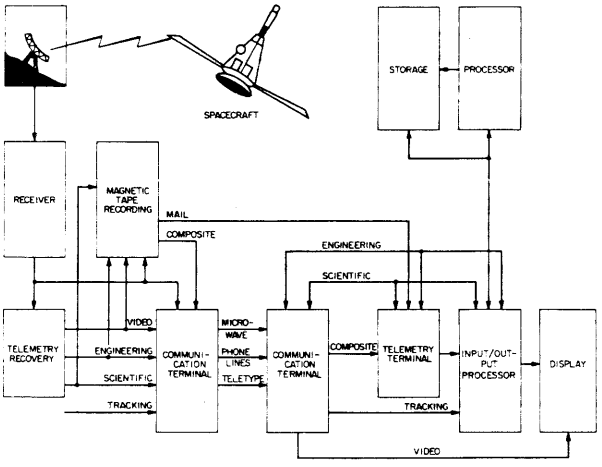


Figure 1. General Data Flow.

2. Provide a form of parallel processing of analysis programs. This processing to be at the millisecond level for restricted classes of computation, such as conversion to engineering units and display, and at the 15-minute level for the major analysis programs such as orbit determination, midcourse guidance analysis, science analysis, and engineering systems analysis.
3. Provide automatic on-line alarm monitoring and distribution of both raw data and reduced data in three analysis centers and at a central operations center. The system must have the capability to automatically retrieve past telemetry and tracking data.
4. Provide capability for distribution of selected parameters to a central status display.
5. Provide processing capability for the complete non-real time processing of all data associated with the spacecraft test program and flight.

The advantages of this approach are:

1. Providing an independent computer for input/output distribution and conversion leads to a natural subdivision and simplification in the programming systems effort. A basic monitor for the sequencing of the analysis routines has been developed placing very few restrictions on the independent development of analysis programs. The input/output processor programming system can similarly be designed to efficiently handle the large volume asynchronous input/output problem. This was a major advantage at JPL due to the sophistication and magnitude of the analysis programs.
2. The system provides the ability for independent expansion of input/output processor, bulk storage, and main processor.
3. The system provides the capability for using only the input/output processor during phases of the mission where only acquisition, monitoring, and non-real time batch processing are required. This frees the larger more expensive main processor for general scientific computing. These same general advantages were later noted in a paper by W. Bauer.[1]
4. The above advantages assisted in more optimum selection of computers. The input/output processor was selected for high input/output capability and byte logic, while the main processor was selected for arithmetic capability and speed of double precision operations.

II. HARDWARE SYSTEM

The Hardware System is composed of three major subsystems: the Central Computing Complex, the Telemetry Processing System, and the Command, Control and Display System.

A. *Central Computing Complex*

The central computing complex consists of three major subsystems: an IBM 7094, an IBM 7040, an IBM 1301 disc file system, and a Direct Data Connection between the IBM 7040 and the IBM 7094. The initial configuration, scheduled to be operational on January 1, 1964, will have a second IBM 7040 for backup. The second configuration, scheduled to be operational on January 1, 1965, will have all three systems backed up with a full dual thread system.

A graphic description of the system configuration is shown in Figure 2.

1. IBM 7094—Main Processor

   The composition of the IBM 7094 system is shown in Tables I and IV. A detailed description of the IBM 7094 may be found in IBM publications A22-6703 and G22-6647.

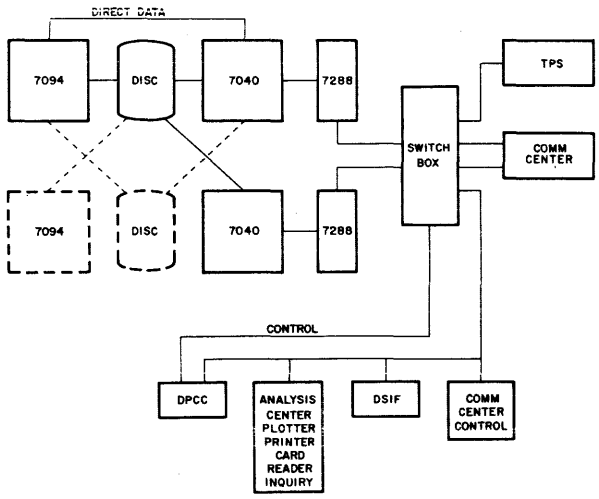


Figure 2. Hardware Configuration.

No.	UNIT No.	NAME	REMARKS
	7100	CENTRAL PROCESSING UNIT	
	7606	MULTIPLEXER	
	7302	MEMORY	32K WORDS
	7909 II	DATA CHANNEL	COMMUNICATE WITH 1301 DISC FILE
	7607 II	DATA CHANNEL	8-TAPE DRIVES AND DIRECT DATA CONNECTION TO THE IBM 7040
	7607 I	DATA CHANNEL	8-TAPE DRIVES AND MONITORING GROUP
	716	PRINTER	
	711	CARD READER	MONITORING GROUP
	721	CARD PUNCH	
1	7631	FILE CONTROL	
2	1301	DISC FILES	
16	729 IV	TAPE DRIVES	

Table I. 7094 Configuration.

No.	UNIT No.	NAME	REMARKS
1	7106 IV	CENTRAL PROCESSING UNIT	32K-WORD CAPACITY
2	7288	DATA COMMUNICATIONS CHANNEL	
1	1414 IV	I/O SYNCHRONIZER	USED FOR MONITORING, TEST, etc.
1	1402 II	CARD READER PUNCH	
1	1403 II	PRINTER	
1	7904 II	DATA CHANNEL	SERVICES CORE AND TAPE CONTROL AS WELL AS DIRECT DATA CONNECTION
1	1414 I	I/O SYNCHRONIZER	SERVICES 8-TAPE DRIVES
8	729 IV	TAPE DRIVES	TAPES PRESENTLY ASSIGNED

Table III. 7040 System Configuration.

The IBM 7094 system is a high-speed digital data processing system. In the SFOF digital data system it serves as the primary element for the processing of the complex analysis programs, such as the Orbit Determination, Tracking Data Editing, and Guidance Maneuver programs. These programs, because of their size, may require the entire capability of the 7094. All programs and data are stored on the disc and are transferred to the 7094 when required. Results in the form of tabulations, plots, commands and antenna pointing predictions are returned to the disc when completed.

2. IBM 7040—Input/Output Processor
   a. IBM 7040 Central Processing Unit

      The composition of each IBM 7040 system is shown in Tables II and III. A detailed description of the 7040 system may be found in IBM publication A22-6649-2.

No.	UNIT No.	NAME	REMARKS
1	3880	EXTENDED PERFORMANCE	EXTENDS DOUBLE PRECISION AND REFERENCE FLOATING POINT
1	5080	MEMORY PROTECT	
1	7498	STORAGE CLOCK & INTERVAL TIMER	
1	W04882	DIRECT DATA ON 7904	
1	W05187	FILE PROTECT INDICATOR	ALLOWS TESTING FOR FILE PROTECT PRIOR TO WRITE SELECTING A TAPE

Table II. 7040 Special Options.

Two IBM 7040's are being incorporated into the initial configuration of the SFOF Central Computing Complex to provide the degree of reliability required for the space flight operations. One is designated the prime IBM 7040; the second serves as backup. The two IBM 7040 systems are identical in all respects. In the event of a failure of the prime system, switching arrangements are provided to remove the prime system from on-line operation and substitute the backup system. The IBM 7040 performs the following functions.

(1) Input Processor. All inputs to the Central Computing Complex are routed through the IBM 7040. These inputs are from the communications interface and from the inquiry con-

No.	UNIT No.	NAME	REMARKS
1	W05229	STORAGE CELL CLOCK & INTERVAL TIMER	
1	W05186	INTERVAL TRAP CONTROL	
1	W04882	DIRECT DATA CONNECTION	ON 7607 II DATA CHANNEL
1	W05664	2-WAY DIRECT DATA SWITCH	ON 7607 II DATA CHANNEL
1	W05185	TEST READY WRITE	ALLOWS TESTING FOR FILE PROTECT TO WRITE SELECTING A TAPE
1	W04917	TEST READY STATUS	ALLOWS TESTING FOR READY STATUS PRIOR TO READ OR WRITE SELECTING
1	W04221	THREE WAY 7631	ALLOWS DISC TO OPERATE WITH EITHER OF TWO 7040'S

Table IV. 7094 System Special Options.

soles and card readers in the seven remote analysis and control areas. The IBM 7040 formats, time tags, logs and routes this input data to other parts of the system.

(2) Output Processor. The IBM 7040 acts as the source and control of all displays in the seven remote areas. It also formats and controls all output to the teletype and high speed communications nets.

(3) Quick-Look and Alarm Processor. The IBM 7040 decommutates incoming telemetry data, converts this data to engineering units and monitors the data for out-of-tolerance conditions. In response to requests from remote areas, a subset of this data will be printed and/or plotted in the remote area within milliseconds of real time.

b. IBM 7288 Data Communications Channel

An understanding of this channel is important for an understanding of the whole data processing system. A simplified block diagram is shown in Figure 3.

The IBM 7288 Data Communications Channel expands the input/output capabilities of the IBM 7040 computer. The channel can provide direct connection with a variety of external devices operating at various speeds for real time operation. The data link between the IBM 7288 and the external device is 36 bit parallel; however, special adaptors can accommodate serial or parallel transfers of fewer bits. Like other overlapped data channels, the IBM 7288 transfers data to and from the computer core storage in a 37 bit parallel fashion concurrent with operation of the central computer program.

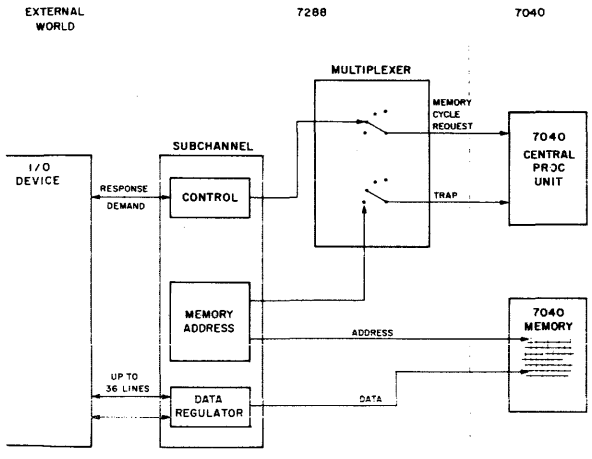


Figure 3. IBM 7288 Data Communications Channel.

The number of input/output connections (subchannels) is limited to 48.

The external equipment can access the computer memory with little supervision by the central processor. The IBM 7288 controls the address of the memory areas to be used for input/output data, and interrupts the IBM 7040 program when the external equipment has filled (input device) or emptied (output device) its core area. Each subchannel has its own unique buffer area in the 7040 core memory. The location and length of this buffer is set by switches and jumper wires within the 7288 subchannel. The buffer location may be assigned to any portion of the core storage, and may be of any length up to 512 words. This assignment is made at the 7288 and may be field changed, but cannot be altered with programming. Theoretically, data transmission between the IBM 7040 and the IBM 7288 can be 62,500 words per second maximum. Practically, the transfer rate will be limited by program interrupt frequency, trap processor program length and loading of the other data channels.

The multiplexor is the control portion of the IBM 7288 to which subchannels for input/output devices are added. It contains control circuitry for determining in what

priority subchannel service requests are granted. There are two separate priority networks: one which controls subchannel memory cycle requests, and one which controls subchannel program interrupt requests. These two priorities are assigned independently, so that it would be possible for a subchannel to have high memory cycle priority but low program interrupt priority. Memory cycle requests always have priority over program interrupt requests.

The subchannels do the interfacing between external equipment and the multiplexor. They contain a data register for one word of data, the circuitry for demand-response control of the external device to subchannel data transfer, control circuitry for giving memory cycle and program interrupt service requests to the multiplexor, and a memory address register to control which 7040 core location a word of data will reference.

For an input subchannel, the sequence of events is as follows: (1) the IBM 7040 program selects the subchannel, thereby allowing the subchannel to function, (2) the input device signals the subchannel that there is data to be sampled on the data lines, (3) the data is entered into the subchannel data register and the subchannel initiates a memory cycle request, (4) the multiplexor grants the memory cycle request, the data word is entered into the 7040 core location designated in the subchannel memory address register, and this memory address register is incremented by one, (5) the subchannel signals the external device that it is ready to receive another data word, (6) steps 2 through 5 are repeated until the block of core storage assigned for the subchannel is full, (7) the subchannel initiates a program interrupt request to the multiplexor, (8) the multiplexor initiates a data channel trap to the CPU. Steps 2 through 8 continue until the computer program or a non-standard occurrence in the device causes the subchannel to be deselected, thereby inhibiting further data transfer. By reversing the memory cycle and program interrupt logic, and the logic for demand-response control between the subchannel and the external device, the subchannel becomes an output subchannel. Table V is a list of subchannels planned for the initial computer configuration.

3. Disc Files

   The disc file system consists of an IBM 7631 File Control and two IBM 1301 Disc Files. The 7631 can communicate with one IBM 7040 and one IBM 7094 on a time shared basis. Each IBM 1301 can store 54 million characters, to give the system a total medium access storage of 108 million characters.

   The disc file is the volume data storage medium in the computing system. The disc file will contain the disc dictionary, all raw data, all 7094 generated output data, the raw data table, the master data table, the operating programs for both the IBM 7040 and the IBM 7094, planetary and lunar ephemerides, and scratch regions to be used by analysis programs.

4. Direct Data Connection

   The Direct Data Connection allows the parallel transfer of 36 data bits or 10 sense bits between the 7904II Data Channel on the IBM 7040 and the 7607II Data Channel on the IBM 7094. Either the IBM 7040 or the IBM 7094 programs may initiate the transfer. Transfer rates are limited only by the core memory speed of the IBM 7040 and can reach instantaneous speeds of 62,500 words per second.

   The Direct Data Connection is used primarily to transfer control information between the IBM 7040 executive program and the IBM 7094 executive program, and between a remote anal-

ysis area and its associated IBM 7094 analysis program. The most important function performed is the control of joint disc file usage. When either of the computers wishes to have access to the disc file, the other computer is notified via the Direct Data Channel.

5. Computer Complex Switching

There are two ways that the required reliability of the data processing system is achieved. The first method is by providing subsystem redundancy with the capability of switching in backup subsystems. This capability is described in this section and in Paragraph II. C. 5 below. The second method is by planning alternate operational modes to allow continuing operation at reduced capability in the event of subsystem failure. These modes are discussed in Section IV. B.

The computer complex switching will be implemented in two steps. The first, to be available in the initial system operational on January 1, 1964, will allow the disc files to be switched to either of the two IBM 7040 systems and the Direct Data Connection to be switched from the IBM 7094 to either of the IBM 7040's.

The second step will be accomplished when the second disc file system and IBM 7094 are added to the SFOF Data Processing System by January 1, 1965. In this configuration, it will be possible to switch either IBM 7040 to either disc file, and either disc file to either IBM 7094. It will also be possible to switch the Direct Data Connection from either IBM 7040 to either IBM 7094.

The initial system will be able to recover from any failure in the IBM 7040 system without loss of capability. The single thread computer complex (IBM 7040-Disc file-IBM 7094) is estimated to have a mean time to failure of 55 hours. With the IBM 7040 backup, the initial system will have an estimated mean time to failure of the data acquisition, quick-look and alarm monitoring functions of 10,000 hours.

In the 1965 configuration, it can be seen that it will be possible to have two full single thread computing complexes, and that no capability will be lost unless there is simultaneous failure of two IBM 7040's, two disc file systems, or two IBM 7094's.

6. Volume Output

Large volume data output will be prepared off-line by using an IBM 1401 system for tabulations, and a General Dynamics Electronics 4020 microfilm

	No. OF UNITS	No. OF SUB-CHANNELS	MULTIPLEXING RATIO	40-CORE BUFFER LENGTH WORD/CHANNEL	SUBCHANNEL-UNIT TRANSFER BITS	SPEED
TTY IN	12	2	6	40	1	
TTY OUT	12	2	6	100	1	
PHONE IN	2	2	—	100	36	
PHONE OUT	1	2	—	200	36	
μ WAVE IN	1	1	—	100	36	

	No. OF UNITS	No. OF SUB-CHANNELS	MULTIPLEXING RATIO	40-CORE BUFFER LENGTH WORD/CHANNEL	SUBCHANNEL-UNIT TRANSFER BITS	SPEED
ADMIN PRINTER	8	1	32	100	6	300 ch 's 4-5 lines s
INQUIRY	7	1	20	14	36	10K w s
STATUS		1		8	35	10K w s
PLOTTER A	5	5	—	100	35	10 pts 's
PLOTTER B	4	4	—	100	35	5 pts 's
PRINTER (TAB)	5	5	—	200	6	500 c s
CARD READER	5	1	8	80	6	200 cards min

	No. OF UNITS	No. OF SUB-CHANNELS	MULTIPLEXING RATIO	40-CORE BUFFER LENGTH WORD/CHANNEL	SUBCHANNEL-UNIT TRANSFER BITS	SPEED
CLOCK	1	1	—	1	36	
INTERVAL TIMER	1	1	—	1	1	
SENSE (COMM)	1	1	—	12	36	

Table V. Subchannels.

printer/plotter system for both plots and tabulations.

a. General Dynamics Electronics 4020

The function of the GDE 4020 is to produce at high speed large volume tabular printouts and plots. Magnetic tape output from the IBM 7094 system is carried to the GDE 4020 system for processing. Output from the GDE 4020 is in the form of 35 mm film or exposed 7½ inch by 7½ inch paper, known as the hard copy option. The film is processed through an automatic Fulton processor and hard copy is provided using a Haloid Xerox Copy Flo machine. The hard copy option output must be processed through a standard oscillogram processor. Microfilm output can be converted to hard copy in approximately one-half hour. The hard copy option can be developed in 10 minutes (Ref: General Dynamics Electronics, Spec. II).

b. IBM 1401 System

The two IBM 1401 systems produce large volume tabular printout and punched cards. Each IBM 1401 system consists of an IBM 1401 central processor, an IBM 1402 card reader-punch unit, an IBM 1403 printer, and two IBM 729-II tape units. The printer is a 600 line-per-minute printer with 132 characters per line (Ref: IBM manual D24-1401-1). Tabular output is available within five minutes of completing the writing of the magnetic tape output on the IBM 7094.

B. *Telemetry Processing System* (Figure 4)

The purpose of the Telemetry Processing System (TPS) is to convert telemetry data received in analog, digital, or composite sub-carrier form to IBM 7288 subchannel compatible, 36 bit format or to IBM compatible magnetic tape. This process is accomplished either in real time, using data from the microwave link to the Goldstone tracking site, telephone lines from the Goldstone tracking site, or telephone lines from the Atlantic Missile Range; or in non-real time, using data recorded on magnetic tapes. The capability for producing strip chart recordings of the analog discriminator outputs exists at both the TPS and in the remote analysis centers.

During the most critical portions of a mission, it is possible to provide parallel processing through TPS, thus providing a backup capability in the event of prime station failure. Each station is equipped with two parallel output buffers which feed subchannels in two different IBM 7288's. In the event of prime IBM 7040 failure, the two parallel output buffers permit the backup IBM 7040 to be switched into the prime position without disturbing the data flow.

The telemetry processing complex (Figure 4) consists of three identical groups of general purpose equipment, and six groups of mission oriented special purpose equipment; two for each of three missions. Patching between one of the special purpose equipment groups and a general purpose equipment group creates a station capable of supporting a given mission. Patching of the twin special purpose equipment group to another general purpose station yields the backup signal processing path mentioned earlier.

Each general purpose equipment group contains a longitudinal wideband tape unit, ground instrumentation discriminators, a

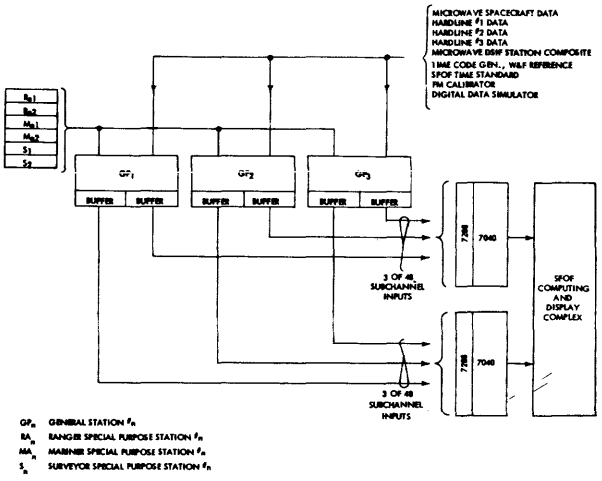


Figure 4. Telemetry Processing System.

time code translator, a tape search unit, a PCM decommutator, an analog to digital converter, and a small general purpose digital computer used as a format converter. The special purpose groups of equipment include mission compatible discriminators, voltage to frequency converters, a PAM decommutator, a bi-phase modulation to digital converter, and all necessary level shifting amplifiers to properly interface between the general and special purpose equipment groups.

The TPS contains a third category of equipment for test, calibration and analysis. This category contains frequency calibration equipment, a panoramic spectrum analyzer, a wave-form generator, and standard electronic test equipment. The Data Analysis Laboratory contains the equipment necessary to do non-standard data recovery that TPS is not able to handle.

C. *Command, Control and Display System*

1. Data Analysis Modules

There are seven areas in the SFOF which have items of computer input/output equipment. These areas perform data analysis and/or command-control functions on the Data Processing System. The equipments described in this section are primarily meant for the data analysis function; however, certain of these modules are located at the two special purpose console areas: the Data Processing Control Console, and the Communications Center Console. Table VI is a listing of the area placement of these on-line input/output devices. A remote analysis area is shown in Figure 5.

	SPACECRAFT PERFORMANCE ANALYSIS AREA (SPAA)	SPACE SCIENCE ANALYSIS AREA (SSAA)	FLIGHT PATH ANALYSIS AREA (FPAA)	OPERATIONS AREA	COMMUNICATIONS CENTER	DSIF	DATA PROCESSING CONTROL CONSOLE (DPCC)
I/O CONSOLE	1	1	1	1	1	1	1
SC 3070 PRINTER	1	1	1	1			1
MILGO PLOTTER	1	1	1	1		1	
DYMEC PLOTTER	1	1	1	1			
CARD READERS	1	1	1	1			1

There are two administrative printers in the Data Processing Control Console I O Console.

Table VI. Remote Center Input/Output Equipment.

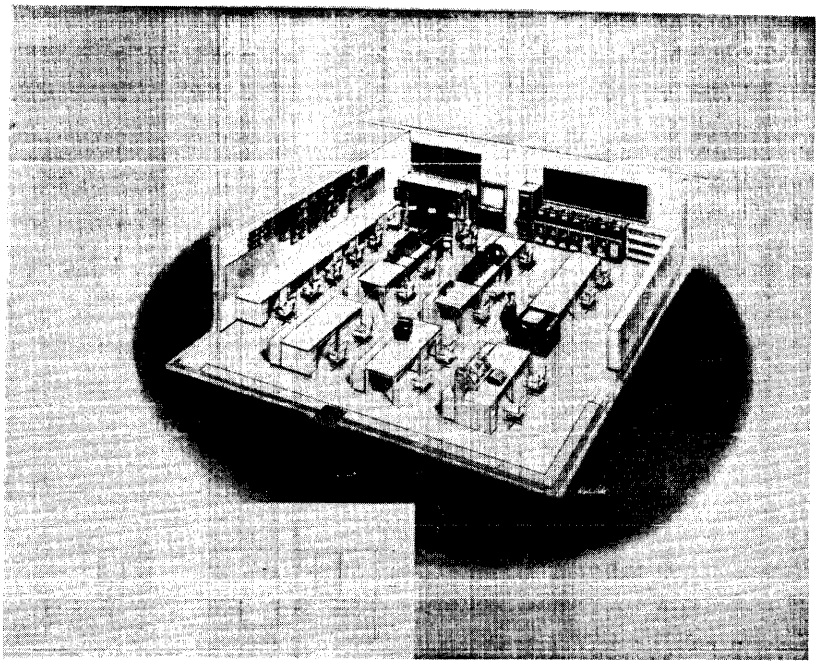


Figure 5. Remote Analysis Area.

a. I/O Console

The I/O consoles will incorporate voice communication modules and modules that display the state of the IBM 7094 analysis program option switches and the name of the analysis program which is currently being processed by the IBM 7094. Three input/output devices will be used to send information to, and receive information from, the I/O consoles.

(1) Inquiry device. This device is composed of an alphanumeric keyboard and 72 character message composer, 36 program option keys, and 25 special option buttons. All inquiry devices are multiplexed into one IBM 7288 36 bit parallel input subchannel with a once-requested-eventually-served priority scheme.

(2) Status Displays. These displays are driven by a 36 bit parallel output subchannel. On each I/O console are eight alphanumeric indicators, and 36 program option lights. The alphanumeric indicators display the current operating program in the IBM 7094 and the analysis program to which the displayed 36 program option lights refer. The display decoder and driver located in each I/O console can be easily expanded to drive additional displays as they become necessary. All status displays are driven by one IBM 7288 subchannel with the update priority under IBM 7040 program control.

(3) Administrative Printer. This is a Motorola TP-3000 electrostatic character printer. The printer will communicate messages to the console as required for feedback from the analysis program or the programming system, and to communicate messages of an administrative nature to the various I/O console areas. All administrative printers are driven by one IBM 7288 6 bit parallel output subchannel with priority under control of the IBM 7040 program.

The I/O consoles will permit the following general functions to be performed from the console location.

(a) Input such things as parameter values, option selection, output format selection, etc., to the analysis programs.

(b) Make requests for on-line plots and listings.

(c) Make requests for off-line IBM 1403 or GDE 4020 plots and listings.

(d) Make requests for encoding and transmission of data to the DSIF sites (commands, predictions, etc.).

(e) Make requests for encoding and transmission of selected parameters to the central status display, or to another analysis area.

(f) Receive messages concerning data condition, processing status, existing operational priorities, etc.

b. General Dynamics Electronics 3070 Printer

This printer is an electrostatic character printer that prints at a rate of 300 characters per second. It can print up to 120 characters per line. It produces a single copy that may be reproduced by Thermofax or other copy reproducing devices. This printer will be used to produce all on-line tabulations of raw and reduced data.

Each GDE 3070 printer is driven by a 6 bit parallel output subchannel of the IBM 7288.

c. Milgo Plotter

This is a 30 inch by 30 inch plot board that has the capability of both ink line or character (point) plotting. It has an annotating head with the full Fortran character set. Paper must be changed manually after each plot is completed. The plotter will be used to plot on-line raw and reduced data. Programming options permit dividing the plotting surface in up to 12 fields for the plotting of multiple parameters.

Each Milgo plotter is driven by a standard 36 bit parallel output subchannel of the IBM 7288.

d. Dymec II Plotter

This plotter has an 11 inch by 17 inch plot board and, like the Milgo plotter, can both ink line and character (point) plot. It also has a character head, but it is limited to 16 characters. An automatic chart advance is a feature of this system and allows many plots to be made before paper needs to be changed, and also allows "pseudo" strip chart plotting. This plotter will also be used to plot on-line raw and reduced data. Programming options permit dividing the plotting surface in up to six fields for plotting of multiple parameters.

Each Dymec plotter is driven by a standard 36 bit parallel output subchannel of the IBM 7288.

e. Card Reader

The card reader subsystem is a Burrough's B-122 card reader that has been modified to interface with the IBM 7288. It is a column serial reader which reads 200 cards per minute. Standard IBM Hollerith cards are used. The card reader serves a function similar to the inquiry device of the I/O console, and, in fact, the two devices can act as backup to each other. In general, however, the card reader will be used where there is a larger volume of data to be input, or where input data can meaningfully be selected from a pre-punched set of cards.

All card readers are multiplexed into one IBM 7288 6 bit parallel input subchannel. Card read-in can be initiated either from the analysis area, or by the computer programming system. If the programming system initiates the card read, then the multiplexing priority is under program control; if the card read is initiated at the card reader, the multiplexor operates on a simple sequencing priority.

2. Communications Center Module

The Communications Center is responsible for switching all teletype lines to provide optimum coverage. The standard I/O console functions will be augmented by two ways that will be used to determine the status and configuration of the communications system. The first way is to assign the 25 special function buttons on the inquiry device to allow rapid reply to queries from the computer program as to source and type of any garbled or non-standard teletype message. The second device is an entry device to the card reader subchannel that echoes the teletype switching network, whenever the network is changed.

3. Data Processing Control Console

In addition to a standard I/O console, the Data Processing Console (Figure 6) has two other modules. One module

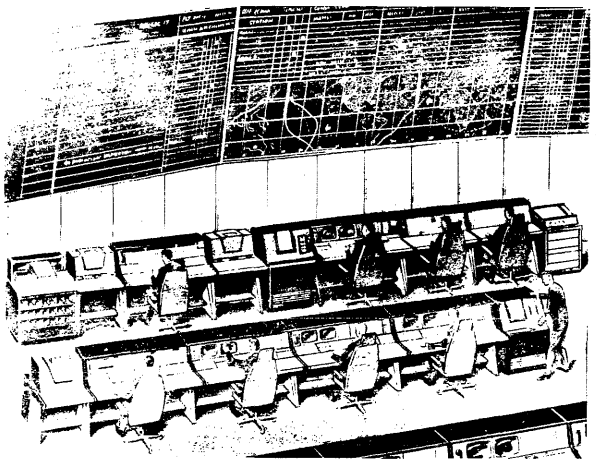


Figure 6. Data Processing Control Console.

contains equipment status indicators for both peripheral devices and central computing complex subsystems, and the controls for effecting the switching as described in Sections III. A. 5. and III. c. 4. The other module contains teletype line switching and status information determined from the Communications Center Console. Controls for initiating "loop" tests between input and output teletype sub-subchannels are also located on this module.

The flow of data through the data processing system is controlled from the Data Processing Control Console (DPCC). All switching of computer complex and input/output equipment is initiated at this console, as well as control of the computer program priorities. To fulfill these general functions, the following specific functions are performed at the DPCC: (1) The equipment status of all computer complex and input/output equipment is displayed, (2) the IBM 7040 and the IBM 7094 program status is monitored, (3) all central computer complex and input/output device switching is controlled, and (4) test routines and diagnostics for all computer complex and input/output equipment are controlled.

4. Data Processing Switching System (Figure 7)

The Data Processing Switching System (DPSS) is made up of three types of modules: switching modules, multiplexing modules, and status generating modules. The switching module switches the data and control lines to or from and input or output device from one IBM 7288 to another. The multiplexing module provides the sequencing or priority of those devices (card readers, inquiry devices, administrative printers and status displays) which time share an IBM 7288 subchannel. The status generating modules convert the contact closures which indicate the status of computer complex equipment, input/output equipment, and teletype line switching into displays for the DPCC. In the case of teletype line switching status, the module also codes and converts this data into a form to be entered, through the card reader subchannel, into the IBM 7040 for use by the 7040 programs.

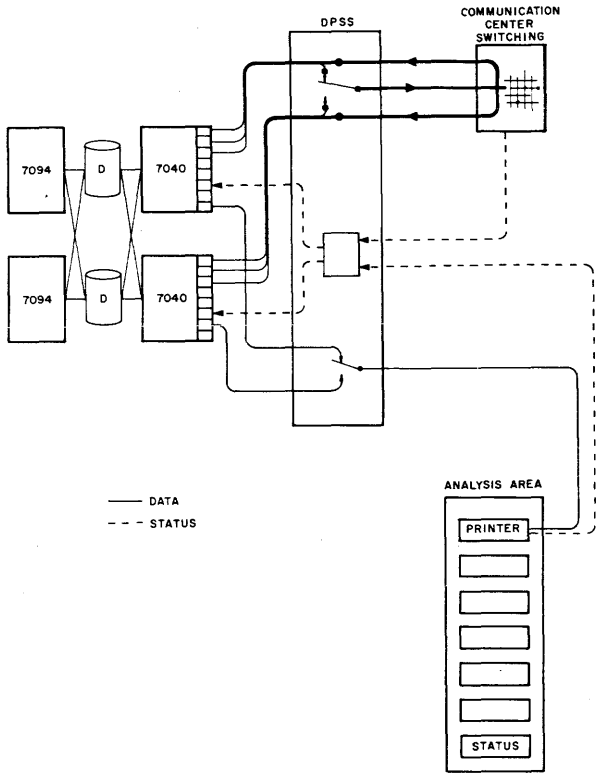


Figure 7. Data Processing Switching System.

5. Timing System

The Data Processing System Timing System is composed of two Astrodata Model 6190 time code generators which have their outputs compared to alarm if one malfunctions. The generators have four types of output: (1) a decimal parallel output for time of day and day of year to one second resolution, (2) pulse outputs at selectable rates from one megacycle to one pulse per hour, (3) a 36 bit parallel binary time of day and day of year with resolution to the millisecond, and (4) a serial NASA 36 bit modulated code output. The timing system can be synchronized with WWV or other suitable Greenwich Mean Time synchronizing source.

The SFOF Data Processing System Timing System has the following func-

tions: (1) drive remote time of day, count-up and count-down displays, (2) enter 36 bit binary time of day into the IBM 7288 clock subchannel, (3) enter pulse rates of one kilocycle, one pulse per second or one pulse per minute into the IBM 7288 interval timer subchannel, and (4) supply the Telemetry Processing System with NASA 36 bit serial modulated time code for time correlating oscillograph and magnetic tape recordings.

III. OPERATING MODES (Figure 8)

A. *Mode I*

This mode will be used where maximum redundancy and minimum failure recovery time are required. It will be used mainly for very critical portions of a flight where the quickest reaction time of the SFOF is required to accomplish the mission.

Mode I involves a full parallel computer complex of two IBM 7040's, two disc file systems, and two IBM 7094's. Data is flowing in parallel through both systems, but only one system is driving display devices and accepting feedback from analysis areas. Very rapid recovery from a malfunctioning system is available in this mode. As discussed earlier, eight-way switching of the central computer complex devices will be available, allowing several sub-modes with different hardware configurations. The backup computing system will be performing all functions of the prime system, except that of display and processing analysis area requests. This mode will not be implemented initially as an on-line backup disc file and IBM 7094 will not be available until 1965.

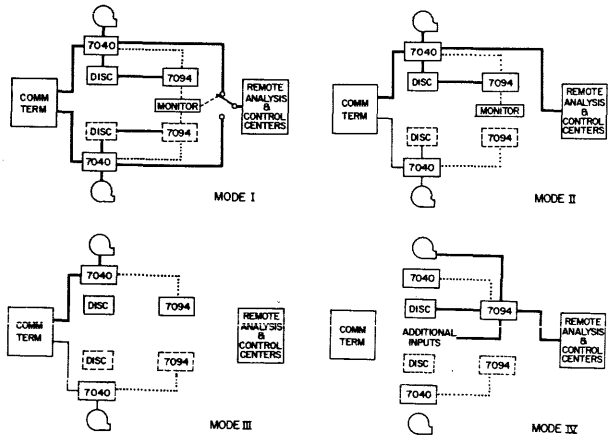


Figure 8. Operating Modes.

B. *Mode II*

1. *Mode IIA*

The input data to the system are flowing in parallel into both IBM 7040 computers. Only one of the IBM 7040's is connected to the user areas and to the disc files and Direct Data Connection. The standby IBM 7040 is logging all input data and preparing a magnetic tape to be used for recovery in the event it is necessary to enter one of the failure modes.

Continuous monitoring of both IBM 7040's as well as the disc and IBM 7094 will be employed so that rapid detection of a failing or failed subsystem can be made. In the event of a major subsystem failure, it will be impossible to maintain operation in this mode, unless the computer configuration of Mode I is available.

2. *Mode IIB*

This mode will provide the same capability and throughput time as Mode IIA, except that the second IBM 7040 and TPS backup will not be on-line with the input data. The throughput time for all types of data in this mode is identical to that of Mode IIA, but failure recovery time will be longer. An IBM 7040 failure could require up to one hour delay in recovery, and a TPS failure could require 15 minutes. As in Mode IIA, the performance of the on-line system is continually monitored to detect existing or impending malfunctions.

In the event of an IBM 7094 or complete disc file failure, it would be impossible to maintain operation in this mode, unless the Mode I computer configuration is available.

C. *Mode III*

In this mode, the IBM 7040's are not connected to either the Disc file system or by the Direct Data Connection to the IBM 7094. Data is being logged for bulk processing on the IBM 7094; quick-look processing, alarm monitoring, and display are

available. This Mode is separated into Modes IIIA and IIIB depending, as in Mode II, if the second IBM 7040 is on-line or not. If the second IBM 7040 is on-line, it will allow a one minute IBM 7040 failure recovery. If the second 7040 is not on-line, up to one hour could be required for failure recovery.

D. *Mode IV*

In this mode, data which has been written on magnetic tape in Mode III will be batch processed by the IBM 7094-disc file system. The processing will be essentially the same as that of Modes I and II, but the input data to the IBM 7094 programs will be on magnetic tape rather than on the disc file.

IV. PROGRAMMING SYSTEM

The overall programming system (Figure 9) logically consists of three different subsystems. Each system operates asynchronously but must be cognizant of the status of the other subsystems. Mutual control between subsystems is required. The three subsystems are: the 7040 system, 7094 system, and the Data Control System which operates in both the 7040 and the 7094 and is responsible for sequencing and issuing all disc file transactions and all direct data communications. The 7040 and 7094 systems both have time sharing schemes; however, these schemes are completely different due to the difference in the functions performed by the two computers.

A. *7040 System*

The 7040 system's function is to handle all raw data inputs, time tag the incoming data with SFOF time, record all data on magnetic tape, transmit data to the disc file, distribute output from the raw data stream and from disc to the analysis centers and to the DSIF and control communications from the remote consoles. The 7040 can also convert data to engineering units and perform out of tolerance alarm monitoring on the telemetry data. The 7040 must be able to perform acquisition and monitoring on two missions in parallel.

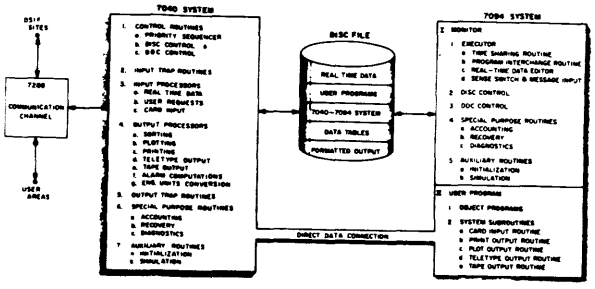


Figure 9. Programming System.

The basic structure of the 7040 includes the following types of routines.

1. Trap Processors
2. Input Processors
3. Output Processors
4. Priority Control

B. *7094 System*

The 7094 system consists of three basic parts: the monitor, the input/output system, and the analysis programs. The analysis routines include orbit determination, science analysis, guidance analysis, etc., and will not be discussed here.

1. Monitor

The monitor is responsible for the control of analysis program execution with a priority scheme, intra-computer communications, editing and sorting of the raw data table to the master data table, status reporting, and accounting and creation of old raw data prints and plots. Since the flight missions divide into logically different operation periods the priority control must be capable of being modified on-line. All modifications of this type are made from the central data processing control console. The 7094 time-shares the analysis programs by keeping a list of the programs which should be sharing time in the 7094. Associated with each program in this list is the percentage of real time which that program should get. An interval timer trap will periodically cause entry to the monitor. The monitor will guarantee the time sharing by interrupting programs, saving them on disc, and starting a new program which needs time.

2. Input-Output Routines

Input routines are necessary in order that an analysis program may request cards to be read from the appropriate analysis center. Output routines provide the analysis program capability to print and plot in the analysis center and to transfer teletype output to the DSIF.

C. *Data Control Program*

Data control programs exist in both the 7040 and 7094. These programs control all disc transactions and all direct data transactions.

The *Control* programs are referenced by the user to obtain disc transmissions. The primary function of these programs is to interpret the user's request and to make appropriate entries into various queue lists. The *trap supervisors* interpret traps from disc and issue new requests by accessing the queue list entries stacked by the control programs. The communication programs provide the necessary intercomputer communication necessary to provide dictionary updating and system control. The *disc routines* are special subroutines used by 7094 trap supervisor and control program.

REFERENCE

1. W. F. BAUER, "Why Multi-Computers?" Datamation, September, 1962.

GROUND OPERATION EQUIPMENT FOR THE ORBITING ASTRONOMICAL OBSERVATORY*

E. J. Habib and A. G. Ferris
Goddard Space Flight Center
Greenbelt, Maryland

H. W. Cooper
Westinghouse Electric Corporation,
Air Arm Division
Baltimore, Maryland

R. L. McConaughy
Grumman Aircraft Engineering Corporation
Bethpage, Long Island, New York

INTRODUCTION

The Orbiting Astronomical Observatory (OAO) is one of a new class of scientific satellites under the cognizance of the Goddard Space Flight Center (GSFC) of the National Aeronautics and Space Administration (NASA). The spacecraft will provide a highly stable and precisely orientable reference for the payload. It will also furnish all the communications, data processing, and power to reorient the observatory, to command the payload, and to store and transmit scientific data to ground stations.

This report presents the ground operation and control aspects of the OAO system. Descriptions of certain of the key portions of the spacecraft system, the operating philosophy, and the experiments carried in the payload are presented to aid in the understanding of the Ground Operation Equipment (GOE) requirements. However, the emphasis in this report is on the Ground Operation Equipment.

THE OAO SYSTEM AND EXPERIMENTS

Figure 1 shows the OAO as it will appear in orbit. The central structure is 10 feet in length and 7 feet in diameter; with the paddles extended it will be approximately 16 feet wide. It will weigh 2500 pounds and will carry 1000 pounds of experimental equipment (for a total weight of 3500 pounds). The flap on top serves two purposes: it is a shutter to protect the telescope optics which will be damaged if the OAO points within a few degrees of the sun, and it is a sun shade to maintain the isothermal environment which will be degraded if the OAO points even within 45 degrees of the sun. With the spacecraft oriented as shown, sunlight arriving from the left side is converted to approximately

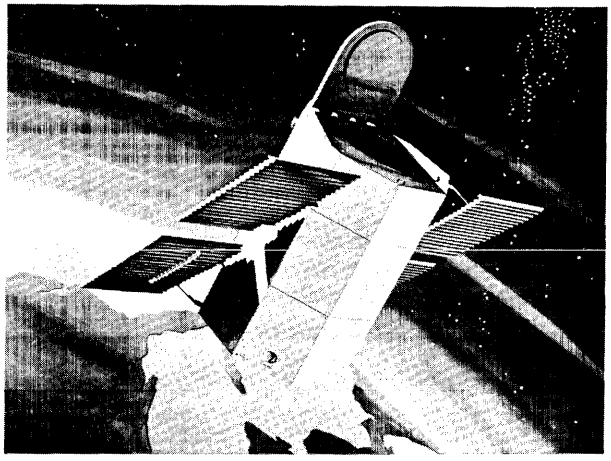


Figure 1. The OAO in orbit.

* Presented at the Institute of the Aerospace Sciences National Tracking and Command of Aerospace Vehicles Symposium, San Francisco, February 19, 1962.

½ kilowatt of electrical power. The mean temperature in the equipment bays is maintained at 10 ± 36°C, while the temperature in the central tube will be isothermal within 10°C of any design temperature between 0°C and −80°C.

The OAO will be placed in a 32-degree orbit at an altitude of 400 to 600 statute miles. In this orbit, the period per circuit of the earth is about 100 minutes and the OAO is within view of a ground station a maximum of 12 minutes. The three remote stations will be located (Figure 2) at Rosman, North Carolina, Quito, Ecuador, and Santiago, Chile. These locations were selected to provide one contact per orbit with a minimum duration of 5 minutes.

The OAO differs from all previous satellites in that it is an orbiting ground controlled scientific laboratory and it is the first to allow extremely precise pointing of the sensors. The control problem is quite difficult. Whereas in prior earth-viewing satellites the means of stabilizing or of procuring a stabilization reference have been simple—either spin stabilzation or stabilization to the earth's vertical with infrared sensors tracking the earth's edge—in the OAO the stabilization to an arbitrary direction in space will be by reference provided by six star trackers. The complexity of the star tracking arrangement requires more sophistication in the ground operation equipment than was required for the previous satellites.

The spacecraft operates in four principal modes: first, the initial orientation and stabilization mode; second, the calibration mode; third, the operational mode; and fourth, the backup, or trouble shooting mode, which is used when there appears to be trouble in the spacecraft subsystems.

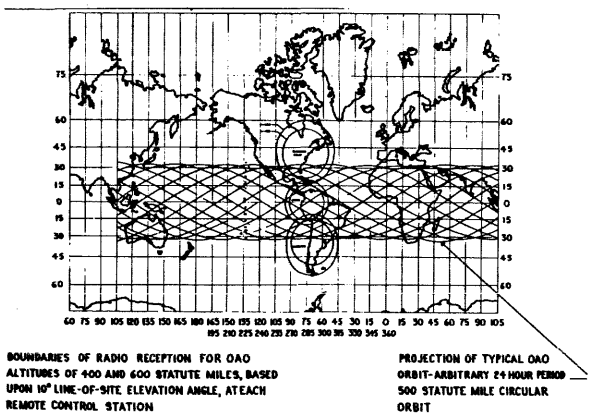


Figure 2. Locations of the remote control stations.

The OAO is designed to stabilize automatically in three axes after being placed in orbit and then to align its optical axis toward a specified area of sky on the basis of information entered in the spacecraft memory before launch. The maneuvers executed by the spacecraft to accomplish this initial orientation and stabilization are described in Appendix A. During this mode, the Gound Operation Equipment monitors the status of these maneuvers and provides a means for assisting in the event of any malfunctions. This mode may take up to three orbits.

In the calibration mode, the OAO's optical devices are calibrated with respect to one another so that the effects of any distortion or inaccuracies which may have been introduced during launch are eliminated.

The major portion of the satellite life will be in the operational mode, gathering data and transmitting these data to the ground control stations.

The backup mode allows the Central Control Station to command the spacecraft in real time through the North Carolina station which is linked to the Central Control Station by microwave.

Figure 3 is a functional diagram of the major elements of the OAO system. In essence, the control equipment on the OAO points the telescope which secures the scientific information; the communications equipment relays these data to the ground stations where they are recorded. The recordings are mailed to the Data Reduction Facility for processing. The scientific data are then distributed to the person conducting the experiment.

The primary experiments for the first three observatory systems are all concerned with stellar astronomy in the ultraviolet range (1000A to 3000A). The OAO-I will carry two prime experiments complementary in their use of the spacecraft systems: (1) a mapping study of the celestial sphere in three ultraviolet ranges (as shown in Figure 4) under the direction of Dr. Fred Whipple of the Smithsonian Astrophysical Observatory, and (2) a broadband photometry study of individual stars and nebulae using the equipment shown in Figure 5 which will be developed by the University of Wisconsin team headed by Dr. Arthur Code. The sky mapping experiment will use primarily picture transmission in real time, while the star

and nebulae study data will be stored and read out on command. In these experiments, the spacecraft controls must aim the roll (or main optical) axis of the satellite at specific portions of the sky with accuracies approaching 20 seconds of arc. Although this is not the ultimate desired accuracy of 0.1 second of arc (corresponding to a displacement of 0.03 inch at a distance of 1 mile), it is near the limit obtainable without using error signals taken from the experimenter's optics.

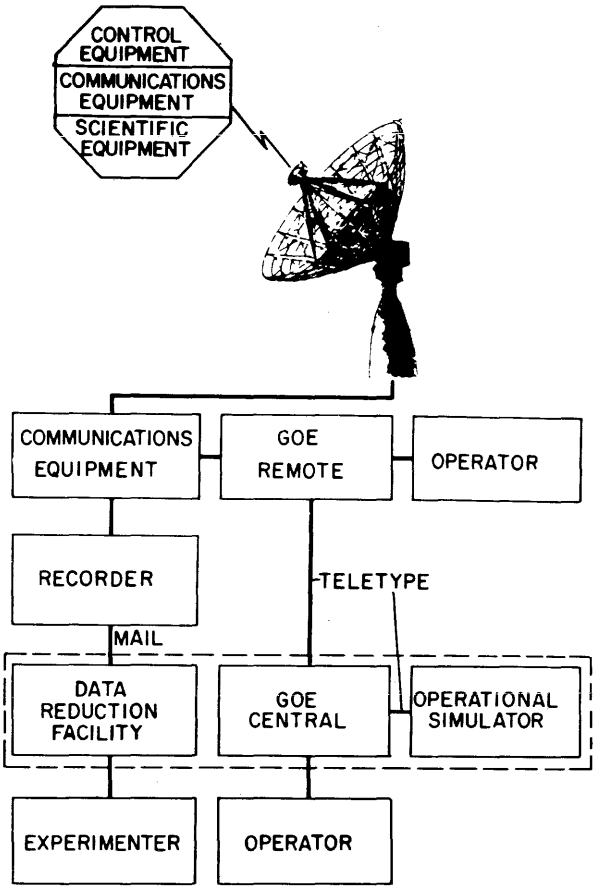


Figure 3. Functional diagram of the OAO system.

The OAO-II will contain an absolute spectrophotometry experiment designed by a GSFC team headed by Dr. James Milligan. The optical system will employ a 36-inch primary mirror and will use both the spacecraft's coarse and fine control capabilities. However, the experiment is designed to obtain useful data even if the fine control system does not operate. Data from this experiment will be stored and read out on command.

The experiment on the OAO-III is concerned primarily with determining the absorption features of the interstellar medium. This experiment will be directed by Princeton University under the direction of Dr. Lyman Spitzer and will require maximum performance from the optical and thermal systems and a control accuracy of 0.1 second of arc. The OAO-IV and later space craft in this series probably will be used for studies of the sun and planets, in addition to a limited amount of payload capacity for small secondary experiments.

In order to accomplish the satellite mission and to secure and transmit the data required by the experiments, the spacecraft requires electronic equipment as shown in the block diagram of Figure 6. The essential features of the spacecraft electronic system are: a precision clock; a core memory data storage system; a TV system for attitude verification; and data process-

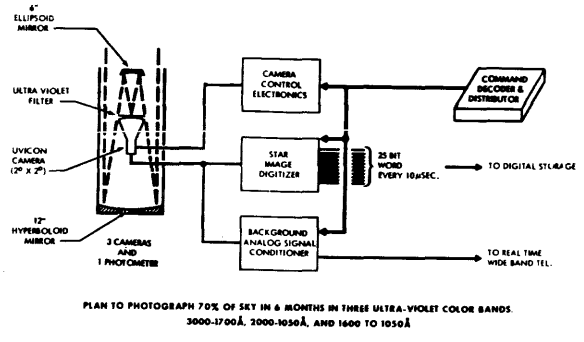


Figure 4. The OAO-I sky mapping experiment: plan to photograph 70 per cent of the sky in 6 months in three ultraviolet color bands—3000A to 1700A, 2000A to 1050A, and 1600A to 1050A.

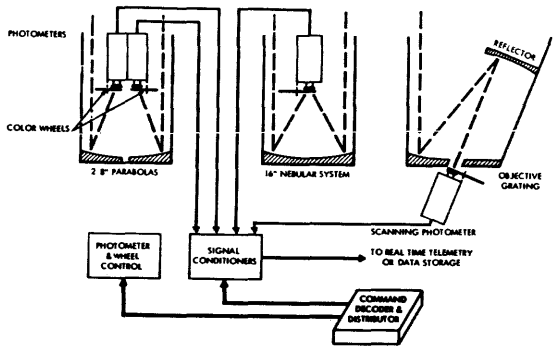


Figure 5. The OAO-I stellar energy exepriment: determination of ultraviolet stellar energy distribution in the spectral region from 300A to 800A.

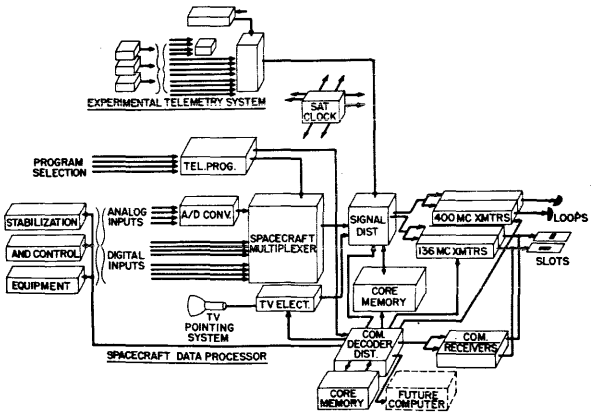


Figure 6. The electronics of the OAO.

ing and communications equipment in the form of transmitters, encoders, receivers, command memory, etc. Provision is made in the electronic system for the addition of a future onboard computer which can reduce the amount of data which must be transmitted from the spacecraft to the ground and back.

The OAO communications include command operation on 148 megacycles and telemetry on 136 and 400 megacycles as shown in the portion of Figure 7 marked "Remote Control Station." The 400-mc receiver is operated at 300-kc IF band width and the narrow-band signal is received on 136 Mc with a 10-kc IF band width. The 300-kc band width is necessary for transmitting the slow-scan television image. The stable nature of the image allows slow-scan techniques with adequate resolution. The Pulse Code Modulation (PCM) data can be transmitted on either the narrow-band or the wide-band links if malfunctions occur in the spacecraft communications.

High-power ground transmitters and high-gain ground antenna systems are used to increase the signal-to-noise ratio. In addition, because of the extreme importance of transmitting correct commands to the spacecraft, the commands and their complements are transmitted, compared and, if correct, retransmitted to the remote control station for further comparison.

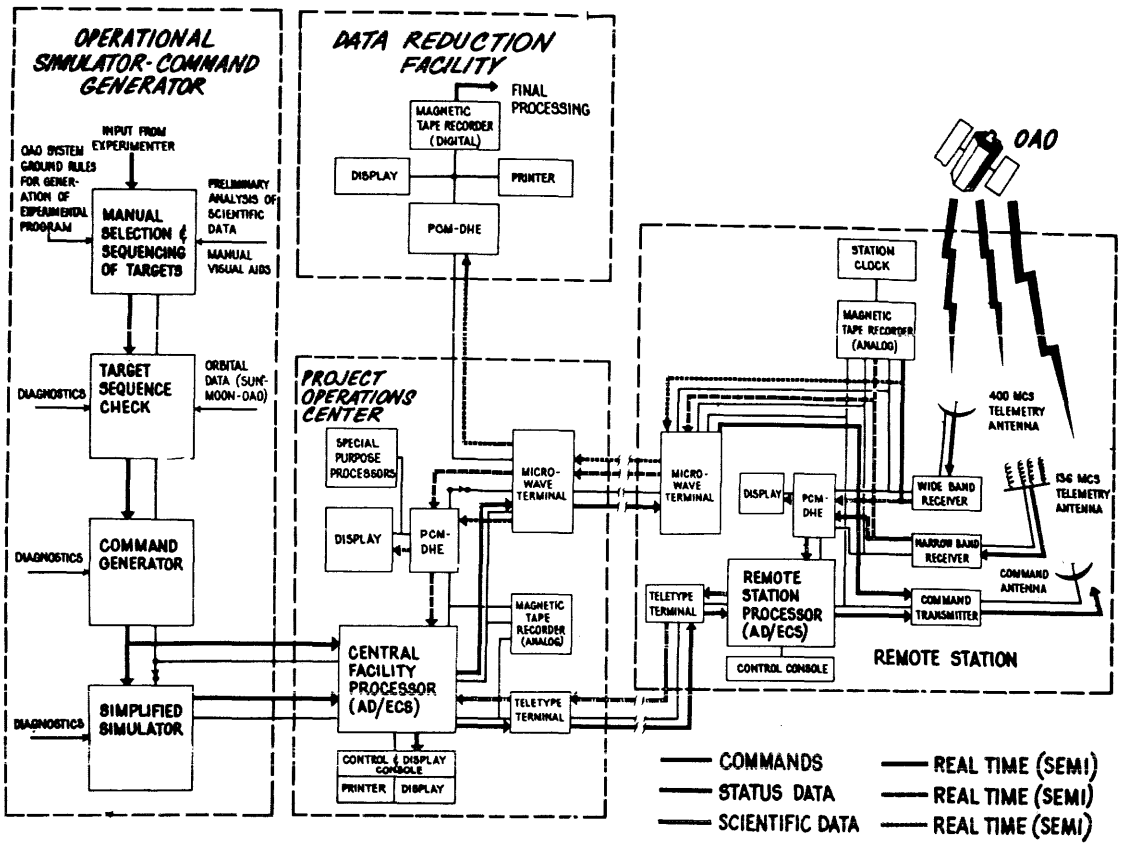


Figure 7. The OAO ground operation equipment.

GROUND OPERATION COMPLEX

Figure 7 is a block diagram of the OAO Ground Operation Equipment complex. Table 1 lists the system elements and functions. The operational simulator-command generator establishes the feasibility of the commands which the experimenter requires. These commands are verified on a large-scale digital computer which introduces orbital data and spacecraft conditions as well as other constraints on the operation. Upon verification, these commands are passed to the Project Operations Center. The commands are then sent by teletype or by microwave to the remote control station which actually communicates with the satellite.

In addition to a radio beacon which is used for tracking, three radio links are used to communciate with the OAO. The remote control stations are all identical in their functions and equipment. They normally only secure data from the satellite and transmit predetermined commands to the satellite. However, in case of emergency or if desired the GSFC Central Control Station can assume real-time control of satellite operations through the North Carolina station. This is done by direct microwave connection. The predetermined commands are originated and checked at GSFC and are transmitted to the remote control stations by teletype at least a day in advance of their use.

The two telemeter links transmit to the remote control stations all the data available in the OAO; these data are recorded on magnetic tape at the remote control stations. Tapes containing scientific or experimental data are mailed to the Data Reduction Facility at the Central Control Station for processing for the experimenter. Data indicating the status of the spacecraft and the experiment are processed by the remote control station—first, to determine whether the satellite is in a satisfactory state to accept commands for the next orbit, and, second, to prepare the data for transmission to the Central Control Station where it may be analyzed and used to up date predictions of the OAO behavior.

The essential functions of the Ground Operation Equipment—message distribution, data processing, signal distribution logic, displays, controls, and command modulators—are diagrammed in Figure 8.

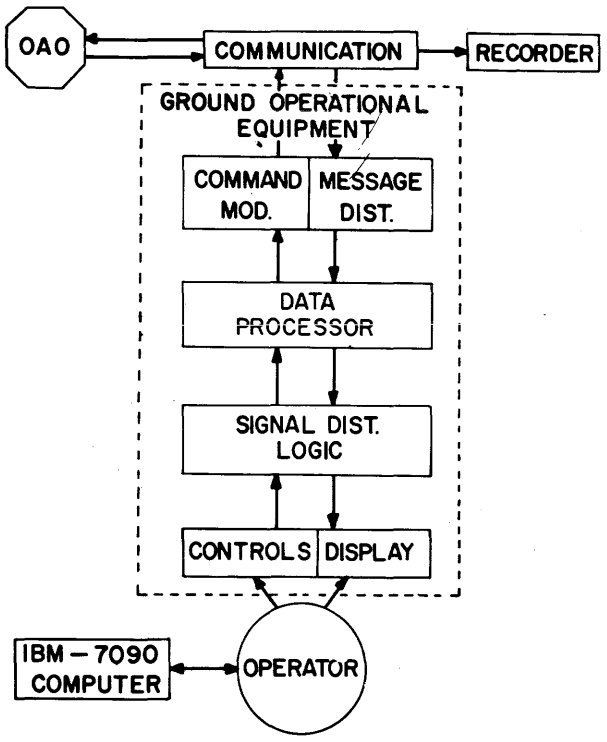


Figure 8. Functional diagram of the ground operation equipment.

The equipment described here will be common to all OAO spacecraft, and portions of it may be used for other future scientific satellites. It monitors the condition of the spacecraft and subsystems and generates the commands necessary to accomplish the scientific experiments and, if possible, to correct any malfunctions that may occur in the satellite.

Remote Control Stations

The data processing function at each of the remote control stations is handled by a general-purpose computer of intermediate capacity, the General Mills AD/ECS-37. This equipment is used in conjunction with a display and control console which provides the operator link to the system (Figures 9, 10 and 11). Figure 12 shows the data flow in the normal mode of operation. The control console at the remote control station (block diagram shown in Figure 13) implements the operating functions which are:

1. Receive commands, predicted status, and instructions from the Central Control Station and store these in AD/ECS-37A computer.

Table 1

System Elements and Functions

Element	Operational Simulator-Command Generator	Project Operations Center	Data Reduction Facility	Remote Station
Equipment	IBM 7090 peripheral equipment.	Communications terminal equipment. PCM-DHE. Display and control console. General Mills AD/ECS-37a.	PCM-DHE. On-off line printer. Off-line plotter. Small computer. Large computer.	RF Equipment. PCM-DHE. Display and control console (limited). General Mills AD/ECS-37a.
Functions	Command Generator Simulator diagnostics.	Duplicate remote control station capability. Display scientific data (limited).	Decomm-sort-catalog scientific data. Quick-look at scientific data. Final Format-output data complete. Status data time. History by parameter.	Command Transmission. Status compare. Command Verification. Format conversion and check. Decomm and conversion for display. Scientific Data. Decomm and display (limited). Signal routing.
Inputs	Orbital data. Star catalog. Sequence of targets. Status of data-spacecraft and experiments.	Punched paper tape from remote control stations. Predicted status from operational simulator. Command from command generator.	Magnetic tape from remote control station. Punched paper tape from project operations center. Direction and Requirements from experiments.	Punched paper tape from project operations center. RF (NB & WB) from spacecraft.
Outputs	Commands. Predicted status. Diagnostics. Ground rules for experimenters.	Punched paper tape to remote control stations. Direct digital to microwave link. Magnetic tape to IBM 7090 Direct to IBM 7090.	Status data-time history. Experimenters data package.	Commands to spacecraft. Punched paper tape to project operations center. Go, no-go to spacecraft system.

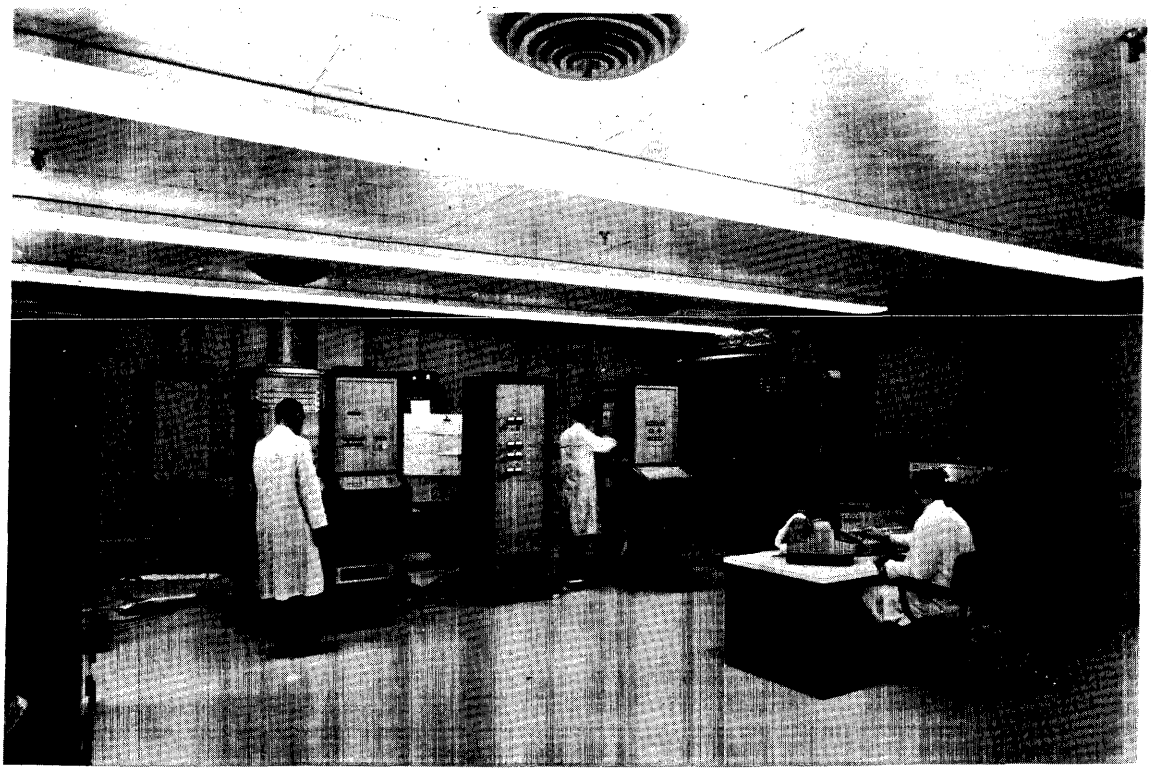


Figure 9. Ground operation equipment at a typical remote control station.

2. Set the Estimated Time of Arrival display per instructions from the Central Control Station.
3. Perform precontact checkout of station equipment.
4. Monitor the Minitrack signal for the OAO contact.

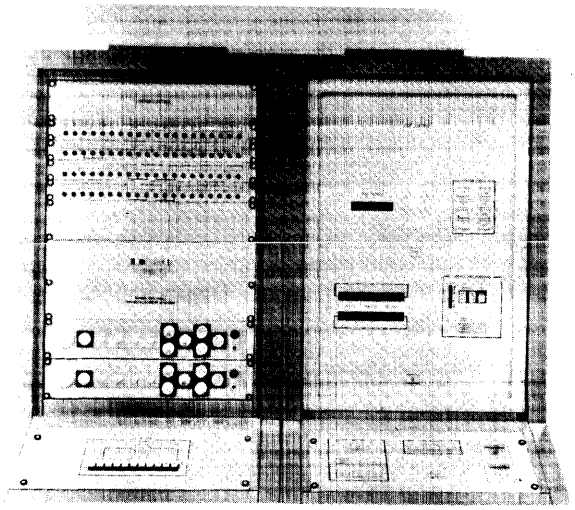


Figure 10. Remote control station operating panel—station control/GOE control.

5. Interrogate the spacecraft when station time agrees with Estimated Time of Arrival and beacon signal exceeds threshold.
6. Verify commands.
7. Receive current status data, check parity, and check with predicted status.
8. If status check fails, transmit hold commands and notify the Central Control Station.
9. Transmit and verify additional commands (stored or manual).
10. Record all commands transmitted and all data received.
11. Display results of command-verify, parity, and status checks.
12. Display Greenwich Mean Time, Estimated Time of Arrival, and real-time contact remaining.
13. Convert status data to TWX format and transmit to the Central Control Station.
14. Send magnetic-tape recording to the Central Control Station.

The remote control station operator may select the satellite transmitters and beacons which he wishes to use and the type of data to

be transmitted from the satellite. The OAO control portion of the console provides this capability (Figure 11). Control over the physical motions of the satellite is limited to preprogrammed commands for future execution or a "hold" command to sustain it in an unchanged condition.

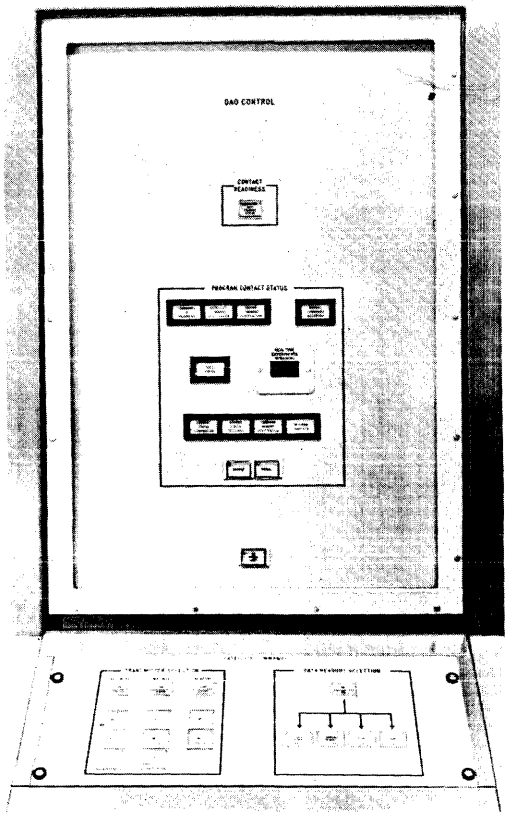


Figure 11. Remote control station operating panel—OAO control.

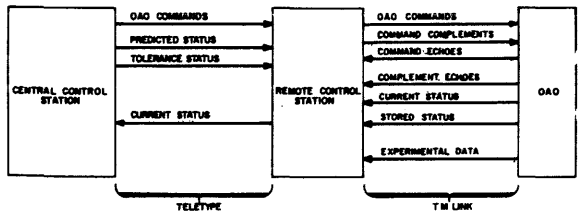


Figure 12. Data flow in the normal mode of operation.

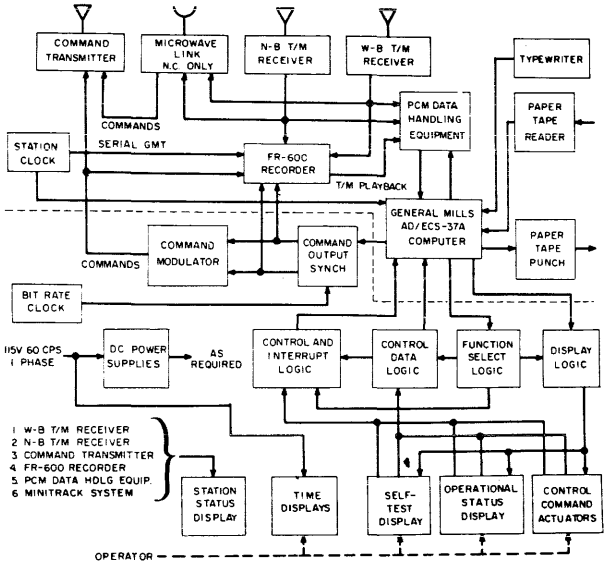


Figure 13. Block diagram of the remote control station control console.

The rest of the console provides control of the ground station and equipment (Figure 10). The station status is a simple go, no-go signal which indicates the readiness of particular pieces of equipment. The system functional test controls initiate actions which establish the operating readiness of the Ground Operation Equipment. The mode selector control is present only on the North Carolina station and transfers control to the Central Control Station via the microwave link.

Central Control Station

The foundation of the OAO ground operations is a large-scale computer which keeps track of astronomical and orbital motions, which maintains an up-to-date status of the observatory, and which accepts programmed operational and performance constraints of the OAO system. In addition to the large scale computer the equipment in the Central Control Station is the same as that at the remote control stations with certain display and control portions augmented. The Central Control Station console consists of five relay racks of equipment. The three units shown in Figure 14 combined with those shown in Figure 10 make up the console. Figure 15 is a block diagram of the Central Control Station equipment not including the large scale digital computer.

The functions of the Central Control Station are:

1. Generate commands, predicted status values, contact times, and instructions.
2. Convert commands and other data to TWX format and transmit to remote control stations.
3. Generate and store alternate program for quick takeover.
4. Receive the previous OAO contact data from the remote control stations and convert these data to IBM 7090 format.
5. Display status data.
6. Enter data and experimenter's instructions into a large-scale digital computer for evaluation and use in generating commands for the next contact.
7. Display Greenwich Mean Time and satellite equivalent time.

By using the information from the satellite and the desired sequence of astronomical observations, the computer generates commands in spacecraft language, predicts the state of the spacecraft at the start of each ground contact, and establishes tolerances within which the prediction is valid. The tolerances established for the go, no-go evaluation must account for the accuracy with which the forecast is made, the accuracy with which the quantity can be measured, and the importance of the quantity to the successful performance of the OAO. During the one contact per orbit, lasting from five to 12 minutes, data must be read from the memory on the spacecraft and new instructions stored in the command memory of the spacecraft. The design problem of the ground operation equipment is basically how most efficiently to transfer information from a multiplicity of sensors in the spacecraft to the human operator on the ground, how to assist him in making decisions, and how best to redirect the necessary controlling actions to the spacecraft subsystem.

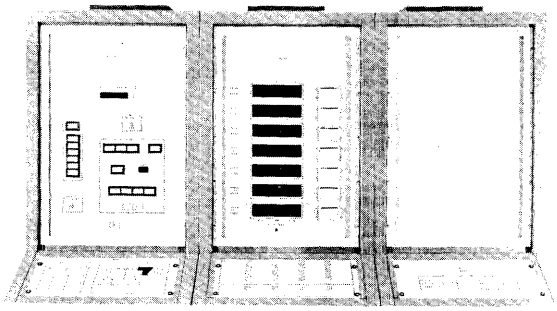


Figure 14. Central Control Station—OAO control/display.

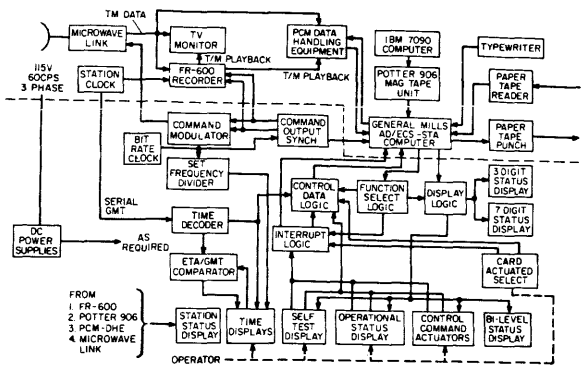


Figure 15. Block diagram of the Central Control Station equipment.

Although the Central Control Station communicates only with the satellite in real time via the North Carolina remote station by microwave link, considerably more data are displayed here than are displayed at the remote control stations.

The Central Control Station equipment enables the operator to examine the data received from 294 analog measurements within the satellite. In addition to the analog status items, there are 192 bi-level status items. These items provide information on equipment components whose status is characterized by one of two states, for example, on-off or open-close. Another 28 items provide, in decimal form, angular data and time from the satellite. The angular data is called up from the AD/ECS-37A Computer using a plastic index card with a title in English which provides a positive means of identifying the display status item. Holes in the card actuate a set of microswitches (Figure 16), which provide coding for entry of the command to the computer, which, in turn, calls up the desired information from the computer memory. For a permanent record of the entire 514 channels, a printer is used.

The optimum amount of information which the operator can read and act upon would require an extensive human factors study. However, the amount of information presented upon the control and display panels at the Central Control Station has been determined by a human factors analysis to be well within the capability of the average operator.

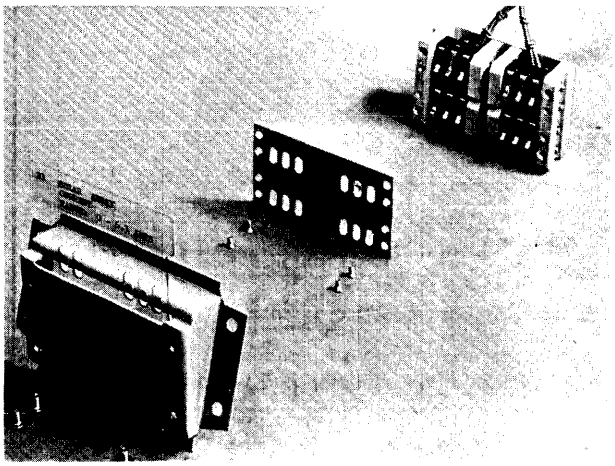


Figure 16. Selector switch assembly.

As a means of verifying the orientation of the OAO, a television camera with a field of view approximately eight degrees square is boresighted along the optical axis of the spacecraft. With a 1-second frame period and the available communications band width, a resolution of 350 lines is possible. The ground based television monitor, provided at the Central Control Station only, can be used for coarse verification of the pointing data derived from the star tracker gimbal angles. The space craft camera, operating in the visible spectrum, is entirely separate from ultraviolet-sensitive cameras provided as part of the experiment.

DESCRIPTION OF OPERATION

Satellite Commands

The most important task of the Ground Operation Equipment is to generate and transmit commands to the spacecraft. The commands transmitted to the satellite consist of seven basic types in either a real-time command mode or a stored command mode. The commands and their codes are given in Table 2.

The Gimbal angle commands are commands to the star trackers, while the attitude change commands control the OAO's momentum wheels directly for changing attitude. Each command consists of two 32-bit words. The first word classifies the command and addresses the proper channel for coding and the second word is the command itself. Figure 17 shows the formats for the control and gimbal angle commands which are typical of command formats. Note

Table 2

Satellite Commands and Their Codes

Command	Code
Control	0001
Data Handling	0011
Address Transfer	0010
Attitude Change	0110
Experimenter's	0111
Gimbal Angle	0101
Ground Synchronization	0100

that the control command format permits 225 bi-level commands.

The commands transmitted to the OAO from the remote stations are in a PCM NRZ format at a 1-kilobit rate. The data transmitted consist of the command words and their complements to provide error checking. The complement form simplifies the on-board command verification. After verification by the electronic equipment in the satellite, the commands and their complements are echoed (retransmitted) to the ground where the ground operation equipment compares the echo from the satellite with the original transmitted command to ensure that they have been properly received.

An alarm is generated and transmitted by the OAO when the complement is not verified on board. Receipt of the alarm causes the Ground Operation Equipment to inhibit transmission of the commands to the OAO. The command

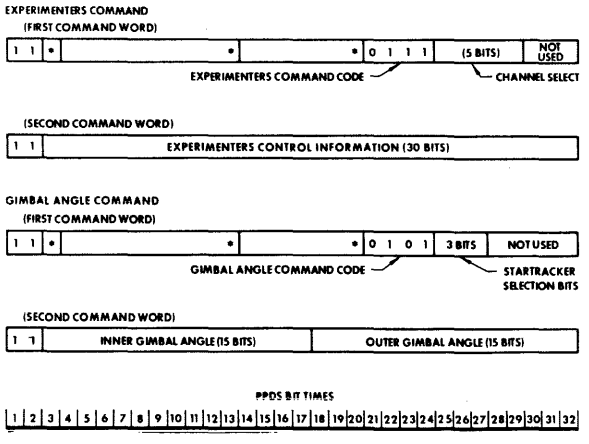


Figure 17. The OAO command format.

message is then retransmitted, beginning with the command that failed, or beginning with a preceding command. Similar action occurs if the command echo is not verified.

Typical Contact with the OAO

Contact with OAO is established at predicted time of contact or when the beacon Automatic Gain Control level indicates that the OAO is in range. The operator initiates a start command to cause the data processor, under control of a stored program, to issue the commands necessary to cause a readout of current status from the OAO. Necessary gating signals are sent to the PCM data handling equipment to define the format and word size for proper signal flow in and from the PCM data handling equipment.

The computer program causes current and predicted status to be compared word-by-word. In the comparison, the limits of predicted status are extracted from storage an item at a time and the difference between limits and actual values for that parameter are computed and evaluated. If one or more items are out of tolerance, the computer causes the spacecraft to be reinterrogated and recompares status. This cycle is repeated until predicted and current status are compared successfully, or until a fixed number of cycles have been executed. If compared successfully, the computer activates a status comparison Go indicator and will sense the position of a Proceed/Halt switch associated with this indicator. If this switch is in the Proceed position, the program will direct the computer to initiate the next subroutine. If the status does not compare or if the switch is in the Halt position, the computer will halt until the Proceed switch or the restart button is depressed. The next subroutine, command transmission and verification, routes commands in the sequence defined by the Central Control Station. The complement of each command word is transmitted immediately following the command word proper as previously described.

Command output synchronizer logic converts the command message to serial form and accomplishes the synchronization and format conversions. The bit-rate synchronizing signal is initiated by computer control and after a delay (3 to 10 milliseconds) determined by the computer subroutine, the command message is initiated. The spacecraft data processing subsystem compares the command word with its complement and also retransmits the command and complement to the remote control station. If the complement check in the spacecraft fails, the spacecraft transmits an alarm in lieu of echoing the command. A gating signal routes the commands through the PCM data handling equipment into the computer where the echoed command is compared with the transmitted command as described previously.

Receipt of the alarm signal initiates a subroutine that will determine which specific command word failed, reset the spacecraft verification logic, and start a new transmission at the command where the error occurred. If this command is transmitted and verified, the process continues until the message is completed or a subsequent error is detected. If a fixed number of repeat cycles has been executed on a given command and an error persists, the New Commands Transmitted light goes red and the program halts until the operator either: (1) starts over, (2) switches in redundant communication links and then starts over, or (3) overrides by depressing the Proceed/Halt switch. When the New Commands XMTD light shows red or no-go, the operator can base his decision of which alternate to pursue by reference to the Satellite Command Verification, Ground Command Verification, and Parity Check lights which indicate the source of the no-go condition.

When the computer has successfully transmitted, received acknowledgment of, and verified the assigned command message it will either halt or proceed with the next subroutine, depending on the state of the Proceed/Halt switch. The manually switched commands may be employed as necessary by the station operator in any of the halt positions discussed previously.

After the contact is completed, the computer is directed to transcribe the actual status data received from the spacecraft to teletype format and to punch a paper tape containing these data. The information on the tape is transmitted to the Central Control Station via teletype. All data received from the OAO (including stored status), all commands transmitted to the OAO, Greenwich Mean Time from the station clock, and an appropriate voice commentary by the OAO operator are recorded on magnetic tape. In certain cases of failure, the Central Control Station may request that stored

status (data gathered during the previous orbit and stored on board) be teletyped back to eliminate the delay encountered in mailing the magnetic tape.

CONCLUSION

The Ground Operation Equipment which has been described and discussed will achieve reliable system operation: First, by forecasting the spacecraft performance on an operational simulator—a large capacity digital computer at the Central Control Station—and establishing tolerances. Second, minimizing human error by using a medium-capacity digital computer as part of the Ground Operation Equipment to compare status data received from the satellite with the forecast status. Third, minimizing human error by displaying a minimum of information to the remote control station operator and minimizing the number of commands that he must or can transmit to the satellite using his own judgment.

A complete status data print out is used in the Central Control Station as a rapid, error-proof means of displaying the greater amount of intelligence available there. With this greater intelligence, emergency control over the satellite may be exercised in a real-time mode. Even here the philosophy is to use this capability as a backup or abnormal operating technique and to rely on the fundamental preprogrammed operational philosophy.

These techniques are new and are fundamental to the success of a precision scientific satellite, such as the OAO, and should find wide application in future satellite and spacecraft upon verification by the OAO program.

APPENDIX A

Initial Stabilization

Immediately after injection of the satellite into orbit, the orientation may be random and there will probably be a residual motion caused by the separation mechanism. There are four steps to accomplish stabilization:

First, the motion is stopped and an initial orientation is established. Three orthogonally mounted rate gyros and six coarse and fine sun sensors accomplish this. The rate gyros detect the motion of the satellite about all axes and, by pulsing gas jets, bring the spacecraft to a near standstill. Simultaneously, the sun sensors locate the sun with respect to the satellite and their signals, also by pulsing the gas jets, rotate the satellite so that the optical axis is aligned with the sun.

Since the sun, in effect, rotates about the satellite at 1 degree per day, the next step is to produce a nonrotating celestial reference in the satellite. This is performed by six highly accurate star trackers gimbaled with respect to the pitch, roll, and yaw axes. The angles of the star trackers with respect to the satellite axis, properly transformed, indicate the pointing of the satellite optical axis.

Before launch six reference stars are chosen, and for that particular day, the angles of these stars with respect to a line connecting the earth and sun are computed. The star trackers are then erected to coincide with these predetermined angles.

The second phase is then initiated by slowly rotating the satellite about the roll axis. At a particular roll angle, the star trackers will produce a star presence signal simultaneously. Since the earth may be occulting some of the stars, a minimum of four simultaneous signals is preferred, although this may be reduced to two on ground command. Once this has occurred, the primary satellite control system, consisting of momentum wheels, is switched from the sun sensors to the star trackers for its error signals.

This provides the initial celestial reference but the optical axis still points at the sun.

The next step is to aim at a desired target star. To accomplish this, the satellite carries core-storage command memory of 256 30-bit words, a star-tracker gimbal-angle digital logic, an analog coordinate transformer, and a system clock.

As soon as the initial reference is acquired, the command memory sends new gimbal angles to the proper star trackers. The large inertia wheels are commanded to rotate through a specified number of revolutions, thus turning the spacecraft through a predetermined angle. While moving to this new position, a star tracker may reach its gimbal limit, but this will be anticipated on the ground and the error signal source will be commanded to another star tracker, again by the output from com-

mand memory. On reaching the new orientation, the star tracker error signals are connected to the small inertia wheels to maintain attitude.

Alternatively, by using the satellite as the reference, the error signal can be used to move a star tracker to acquire a new guide star. By driving either the satellite or the star trackers, a reference in space is continually maintained while moving the satellite to any orientation, subject to the restriction that pointing the optical axis at the sun is not permitted. As a safety measure, the sun shutter automatically closes if the optical axis approaches within 45 degrees of the sun.

Summarizing, the initial stabilization routine is:

1. Initial rate stabilization and solar orientation.
2. Roll orientation.
3. Celestial orientation.
4. Celestial pointing and holding.

ERROR DETECTION CORRECTION AND CONTROL

Robert Steeneck
The Western Union Telegraph Company

Although Error control is a desirable thing to have in any communication system it is obviously much more desirable in communication systems that handle data.

Error control resolves itself into three problems. First, there is the problem of error prevention. Second, there is the problem of error detection, and third, there is the problem of error correction.

Error control in data systems is not a new subject. Considerable effort has been directed toward the control of data errors perhaps long before the invention of the Quill pen.

The advent of digital computers and the associated Data communication systems, however, has now introduced some new sources of errors into data processing.

Consequently considerable effort has been directed in the past few years toward the development of various methods of error control in both computer and communication systems.

As a result of these efforts many methods of controlling digital errors were devised. The use of these control methods, however, has not kept pace with the growth of computers. The reason for this lag lies in the fact that transmission faults are not usually the prime source of data errors.

Humans and machines also contribute errors that are often far more serious than transmission errors. Consequently most data systems usually include error protection methods within their format structure to control this potent source of trouble. Since this type of error protection, originated primarily to control human errors, can also serve to control transmission errors; there is not as much use being made of error control in data communication as one might at first believe to be necessary.

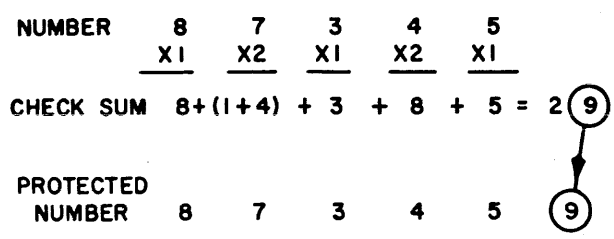
It might be well therefor to examine a few of the methods used to control human errors before investigating the methods used to control communication errors.

Perhaps the first step taken to prevent human errors in the handling of data was to punctuate large numbers in groups of three. Thus a number such as 8734569 becomes much easier to transcribe if it is written as 8,734,569. Today we have progressed to a point where our Social Security numbers are punctuated in a broken rhythm so that the digits read out in a three-two-four beat.

In data handling it is common practice to protect numerical information such as catalog numbers or credit card numbers by the addition of a single check digit. The check digit is usually derived through an arithmetic process involving all of the digits comprising the number to be protected.

The development of a check digit from a simple sum of the digits comprising the number to be protected, however, is not adequate. Such a process affords no protection against the most common of human errors; namely, the transposition of digit pairs during the process of transcription.

Although check systems vary, the protection digit in general is derived by doubling every other digit, adding the results together, and using the last digit of the grand total as the check digit. (Fig. 1)



The Zero, until recently thought to be discovered by the Arabs (actually the zero was discovered by the Mayan Indians five thousand years earlier), not only plays its part in the mathematics involved in data processing, but also is used to protect against loss of information.

Inserting zeros to the left of a number in no way alters the value of that number but does make possible a number format in which all numbers from zero to a preselected maximum contain the same number of digits. By reserving various predetermined specific numbers of digits for various information, a format can be established where the loss of a single digit can be instantly recognized as an error. An example is shown in Fig. 2.

QUANTITY	UNIT PRICE
0 0 0 5	0 0 6 7 2

USE OF ZEROS IN DATA

These are but a few of the many checks and balances used in data handling to guard against errors. Often these methods adequately protect an entire data system against transmission errors as well as data handling errors.

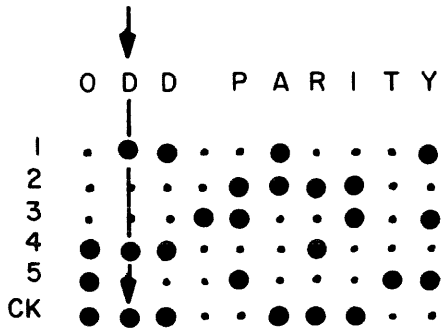
In systems where this protection through format structure does not exist transmission error control would probably be desirable even though the error rate is low. When the error rate is high, however, for example, on overseas H.F. radio circuits automatic error control is vital to successful data communication.

For this purpose Western Union has designed its EDAC equipment to provide automatic error detection and correction to circuits operating at teleprinter speeds. This system will be described later in this paper.

Error Detection

One of the first problems in error control in data communication is that of error detection. Error detection usually requires redundant information applied to each character transmitted or applied to a block of characters after they have been transmitted.

With the advent of digital computers came the first widespread use of parity error detection codes. In parity codes at least one extra bit level is provided exclusively for the purpose of checking the validity each code combination used. The presence or absence of a check level bit serves to make all code combinations contain either an odd number of bits if odd parity is desired, or an even number of bits if even parity is desired. (Fig. 3)



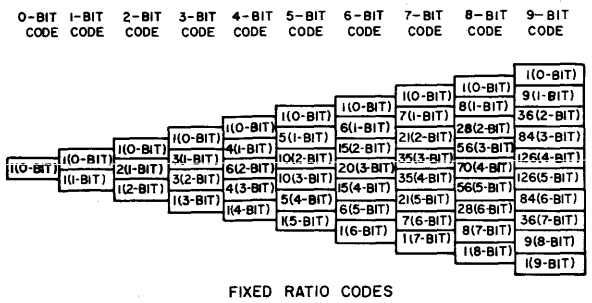
Although a single parity bit included in each character provides adequate protection against undetected errors in computer operation where parallel transmission under more or less locally controlled conditions is used, transmission tests show that at teleprinter speeds the simple parity code fails to detect about 10% of the errors when serial transmission over typical telegraph circuits is employed.

This failure is due to the fact that in serial transmission about 10% of the errors involve an even number of bits.

In some systems error detection is accomplished by using only code combinations in which the selecting- and non-selecting bits always occur in a fixed ratio.

This provides a good means for detecting errors since both gains and losses of an equal number of selecting bits must occur within an errored character to produce an error detection failure.

The chart shown in Fig. 4 illustrates the number of different fixed ratio combinations that are available in various binary codes up to 9 bits in length.



FIXED RATIO CODES

It is interesting to note that the numbers found on this chart exactly duplicate Pascal's triangle of binomial coefficients discovered by this famous French scientist a little over 300 years ago.

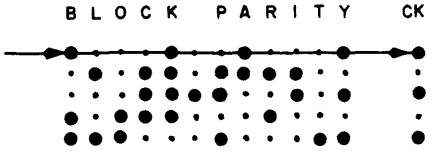
There is undoubtedly some sound mathematical reason that makes the coefficients of the terms obtained when $(X + Y)^5$ is expanded, exactly coincide with the number of 1 bit, 2 bit, 3 bit, 4 bit and 5 bit combinations available in the Baudot code, but the reason is obscure at the present time.

The distribution, however, does suggest that perhaps an improvement in coding efficiency might be obtained by making use of more than one fixed ratio code group when handling data in a given binary code. Different code ratios might then be used to identify the type of data being transmitted.

Whenever error control is obtained through the use of single character error detecting codes, at least one extra bit must be included in every character transmitted.

To improve transmission efficiency block checking systems have been developed in which the check information is added only after a block of information characters has been transmitted.

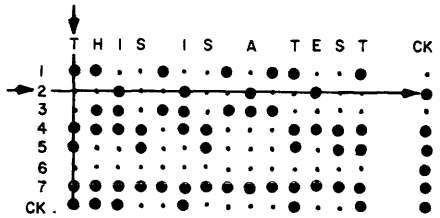
Several block checking systems have been developed, block parity being perhaps the first. (Fig. 5)



In block parity a single character combination is added to the end of each block of information. This added character serves to make the total number of selecting bits in each level either odd or even as desired. In the illustration odd block parity is shown.

When block parity alone is used to protect information it is quite vulnerable to failure whenever serial information is converted to parallel information and directed through circuits individual to each code level. A fault in any of the individual code level circuits will then produce errors that tend to be confined to a particular code level; and under such conditions the chance of detection failure rises to 50%.

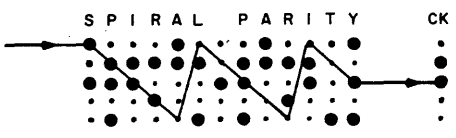
When parity as shown in Fig. 6 is applied to both the individual character and to the block of characters, however, the chance of failure is reduced to an insignificant figure.



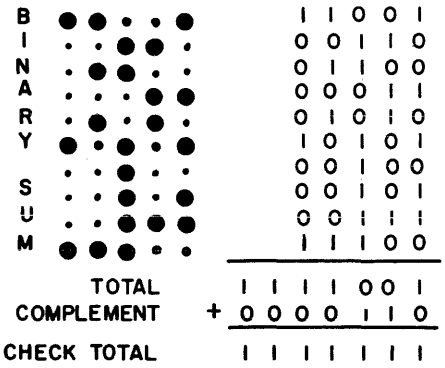
CROSS PARITY

This system of cross parity creates an error detection means that is practically foolproof. It is this system of error detection that is used to control errors in the largest data communication system in the world namely the Air Force Syn-Com Network.

Chances of block parity failure can also be reduced by the use of spiral parity. In spiral parity bits applied at the end of a block of information are not each assigned to one code level throughout the block but are shifted one level for each character transmitted. Errors that occur on one level only are then not apt to influence only one code level of the check parity character and thus the error detection probability is greatly increased. This is shown in Fig. 7.



The shortcomings of simple block parity may also be avoided by considering first of all, that code combinations are in reality nothing more than binary numbers. As binary numbers it is obviously possible to add code combinations together, as illustrated in Fig. 8. The binary number that results may then be transmitted as check information and compared with a similar binary total developed at the receiving terminal.



In actual practice comparison is more easily effected by transmitting the complement of the block binary total. The complement when added to the block binary total at the receiving terminal results in an overall binary total containing no zeros at all. The presence of a single zero thus indicates a non-check condition.

It is not necessary to use the entire binary sum for check purposes although it is quite obvious that the number of bit levels in the binary sum cannot be less than the number of bit levels in the code.

At this point it would be well to analyze the protection to five unit code data that is provided by using only 5 digits of the binary sum as check information.

Whenever transmission erors occur they are generally quite consistent in nature. Most frequently circuits consistently lose information bits; less frequently bits are consistently gained, and the least frequent occurrence of all is the mixed loss and gain of bits of information under the same transmission difficulties.

In the first analysis we will consider the behavior of a five digit check when consistent losses of information bits are experienced. Consistent gains, of course, will produce the same effect.

The basic 5 digit binary counter at a given starting condition when stepped through a cycle of 32 counts or any multiple thereof always returns to that starting point.

In code level #1 each bit has a value of 1 unit in the binary counter and consequently 32 bits in this level would have to be lost in order to lose one complete cycle of the binary counter and thus produce a false check. It would seem therefor that this level has more than adequate protection against lost bits. (Fig. 9)

CODE LEVEL	VALUE	DROPOUTS TO FAIL
1	1	32
2	2	16
3	4	8
4	8	4
5	16	2

ANALYSIS OF USE OF (32 COUNT) 5 DIGIT BINARY SUM AS CHECK INFORMATION

In code level #2, however, the bit value rises to 2 units and the drop-out protection falls to 16 bits, which is still more than adequate.

By the time we examine the fifth level of the code, however, we find that the steady 50% loss in protection for each level has reduced the protection level to only 2 bits. The fifth level thus has only the protection of block parity.

Protection could be improved of course, if more than 5 bits were used for check information, or if a spiral system of binary addition were employed. There is, however, a much simpler solution to this problem.

If the 5 digit binary counter is slightly altered so that it goes through a complete cycle in 31 counts instead of 32 we obtain some unusual results.

The first level with a bit value of one unit now requires 31 drop-outs to produce the loss of a complete cycle and thus produce a detection failure. Its protection level was lowered by 1 bit. (Fig. 10)

CODE LEVEL	VALUE	DROPOUTS TO FAIL
1	1	31
2	2	31
3	4	31
4	8	31
5	16	31

ANALYSIS OF USE OF (31 COUNT) 5 DIGIT BINARY SUM AS CHECK INFORMATION

The second level, however, with a bit value of 2 units will not evenly divide into a 31 unit cycle but will divide into two cycles having a total of 62 units. This results in a dropout protection of 31 bits for this code level as well as for the first code level.

If we examine each code level in turn we find that they all require 31 bit drop-outs for check failure.

Thus, with this simple change in the binary counter error detection protection is equalized on all code levels.

With a 31 bit drop-out protection on all code levels it is safe to say that more than adequate protection is provided for consistent error conditions with this system.

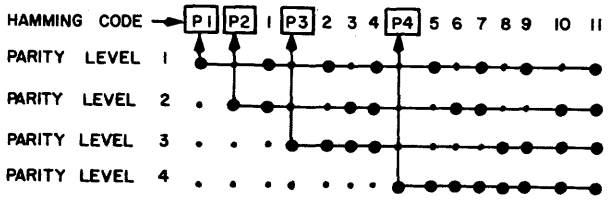
If we examine the system now for operation under the least frequent error condition, namely when both bit losses and bit gains are experienced within the same block, we find that not only are both losses and gains necessary to produce a check failure, but we find that the gains and losses must exactly compensate each other.

Since errors involving both gains and losses under the same transmission conditions are in themselves rare, and since there are many more gain and loss combinations of non-equal value than of equal value the chances of compensation within a block of information are extremely remote.

Error Correcting Codes

Some attempts have been made to design codes that not only detect errors but pin point their location so that correction may be made.

In 1950 Dr. R. W. Hamming suggested a system of coding that could detect and also correct an error if that error involved only one bit of information within the code. (Fig. 11)



HAMMING SINGLE BIT ERROR CORRECTING CODE

The illustration shows 4 parity bits designated as P1, P2, P3 and P4 protecting 11 bits of information designated by the numbers 1 through 11.

The four check bits of the Hamming code each add parity to selected combinations of information bits arranged in four levels in a pattern that resembles consecutive binary numbers.

With this pattern of parity generation any single information bit when errored will produce non-check parity in at least two of the four parity bits of the code.

If we now observe a condition where more than one parity bit does not check and we are reasonably certain that only one bit is in error, the position of that bit may be determined by considering the non-checking parity bits as a four level binary code pointing to the particular single code bit that is in error.

It will be noted that if only one parity bit does not check and only bit has been errored there is only one conclusion possible—the parity bit itself is in error.

The Hamming Code although most ingenious in its concept has not yet found too much use in Data Systems. The reason is that there is no assurance that errors are going to involve only one bit in the code structure.

There is a system contemplated, however, in which bits are first examined upon reception to determine whether any are of doubtful quality.

The Hamming Code will be used in this system and if only one bit of doubtful quality is discovered within a character and a parity failure is observed a bit correction will be made.

If more than one bit of doubtful quality is observed no correction will be made. A re-run of information will be requested.

Other error correcting codes have been proposed in which errors involving more than one bit can be corrected. They require more redundant bits within their structure, however, and also involve more complex error correcting procedures.

It might be noted in the Hamming code that although four parity bits can protect 11 information bits, four parity bits are also required to protect only 5 information bits.

Unfortunately, no error correcting code can function on the complete drop-outs of information that often occurs on overseas H.F. radio circuits. Here error correction can only be effected by retransmission of the errored information.

About ten years ago Hendrick C. A. Van Duuren developed a synchronous system that does just that.

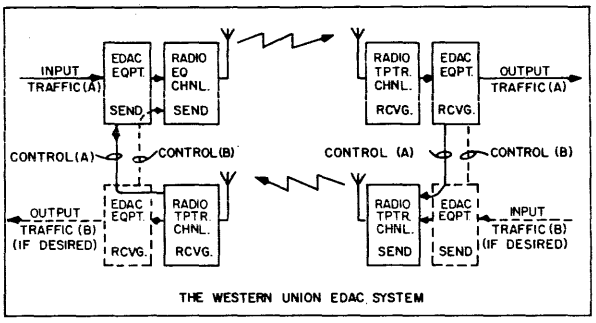
In the Van Duuren system traffic must first be converted into a fixed ratio seven unit code for error detection purposes. Upon detection of an errored character the entire system is

stopped and the errored character repeated from storage.

The accepted information is then converted back to the original code for delivery.

Many versions of the Van Duuren System have been developed, including a Japanese system using an 8 level fixed ratio code. The American version is the well known A.R.Q. system.

Western Union has developed an automatic error detection and correction system known as EDAC and shown in block form in Fig. 12.



THE WESTERN UNION EDAC SYSTEM

In the EDAC system information will be transmitted without code conversion in block form, each block being followed by a single character developed from the sum of the binary bits contained within the block.

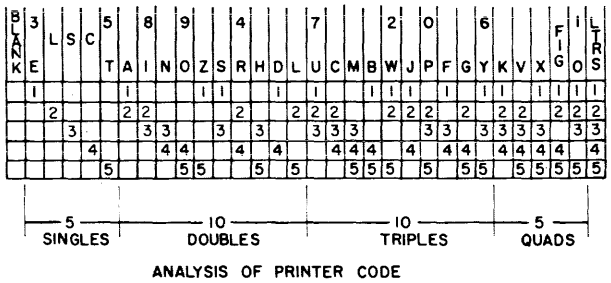
Although transmission will be in start-stop form the character delivery will be paced by a time standard similar to that used on many military circuits.

Pacing the transmission with a time standard not only makes possible the use of enciphering equipment but also provides a simple method of distinguishing between check information and data information at the receiving terminal.

EDAC may be operated on a duplex basis with automatic error control in both directions.

In most data systems numerical information is of far greater importance than alpha information and consequently should receive top priority if only a limited amount of protection is available.

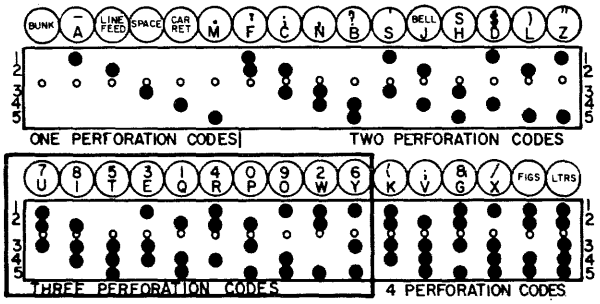
The five unit teleprinter code can provide just this amount of protection within its own structure; for concealed within the five unit code are just ten combinations with a fixed code ratio of three selecting and two non-selecting bits, as shown in Fig. 13.



ANALYSIS OF PRINTER CODE

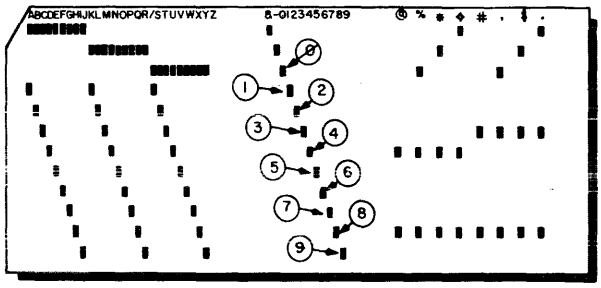
By assigning the ten digits to this fixed ratio code group errors that occur in numerical information will produce meaningless symbols that can be easily recognized as errored numerical data.

Such a code has been used with some German teleprinters and is available to Western Union subscribers who may wish their equipment converted to this code. The code has been christened the L code, L meaning logical. It is the only error control code having zero redundancy. (Fig. 14)



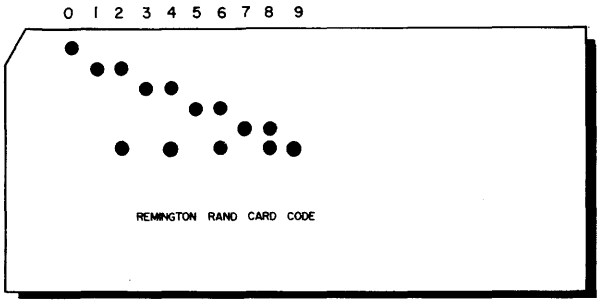
It is interesting to examine various data codes to see if by their structure, they avoid the possibility of conversion through error, of one number into another number.

If we examine the Hollerith card code (Fig. 15) we find that the ten digits are each designated by a single punched hole at different levels in the card. In reading this card code it is thus not possible to convert one number to another number through a reading failure. A failure



to read can only result in the drop out of an entire digit which can easily be recognized as an error.

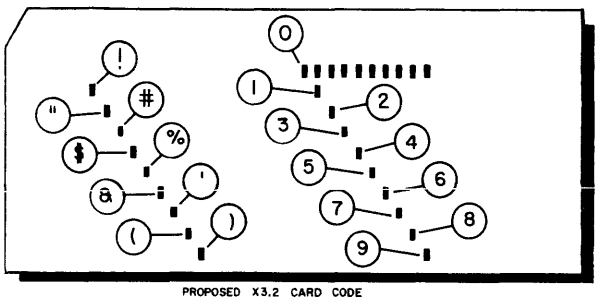
If we examine the Remington Rand Card Code Fig. 16 we find six levels are used for information instead of 12. Six of the ten digits have one punched hole assigned to their code selection and four have two holes assigned to their selection.



Failure to read one of the holes in any of the two hole numeric combinations will always result in the conversion of one digit to another digit, a most serious data system error.

It is perhaps for this reason that the Hollerith Card Code instead of the Remington Rand Card Code is now being used in peripheral equipment of Modern Computers.

It is also interesting to note that the Business Equipment Manufacturers Association is now proposing to change the punched card code so that more information character combinations might be available. To do this they propose to have all digits except zero contain two punched holes as shown in Fig. 17.



A card code of this kind is a step backward as far as error control in numeric information is concerned, as it transfers the tight error control possibilities of the Hollerith card code from numeric information to unimportant and seldom used data symbols. The numeric information then becomes vulnerable to errors that simply alter numerical values of data.

The Business Equipment Manufacturers Association has already adopted an American Standard Code for Information Interchange, the numeric portion of which is shown in Fig. 18.

0 1 1 0 0 0 0 --- 0	0 1 1 0 1 0 1 --- 5
0 1 1 0 0 0 1 --- 1	0 1 1 0 1 1 0 --- 6
0 1 1 0 0 1 0 --- 2	0 1 1 0 1 1 1 --- 7
0 1 1 0 0 1 1 --- 3	0 1 1 1 0 0 0 --- 8
0 1 1 0 1 0 0 --- 4	0 1 1 1 0 0 1 --- 9

This code is similar to the Binary Coded Decimal code used in most computers and is also similar to the Fieldata code used by the armed forces.

This code is quite attractive for computer operation because it provides a simple and more or less direct means for converting numeric information into binary form.

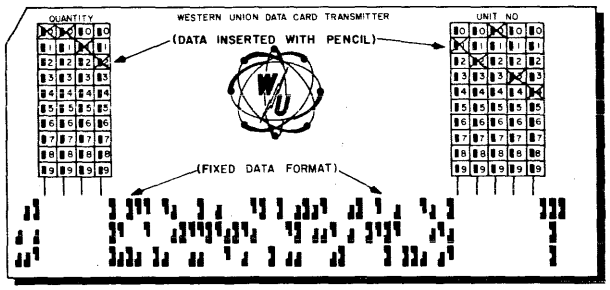
Without error protection, however, this code is most vulnerable to data errors that are most serious in nature for every one of the ten digit code combinations can easily be errored to produce other digits that might be accepted as valid information.

As time progresses we shall see more data prepared automatically without introducing the need for human transcription and the consequent introduction of Human Errors.

Pre-punched cards are now being used in many places to insure the preparation of data messages at the point of origin in proper format without depending on human skills.

In a short time the marked data card developed by Western Union to automate the collection of data at the source will also be available.

The Marked Data card shown in Fig. 19 is printed with conductive ink. Bit markings are printed at the bottom of the card and are read by passing the card under sensing styli. These bit markings serve the same purpose as the punchings in a pre-punched card in that when sensed they are converted into the fixed format text of a data message. The data proper may be inserted in the upper portion of the card by marking appropriate boxes with a soft pencil and thus establishing conductivity to the data boxes selected.



Readout errors and errors due to careless marking in the upper portion of the card are discovered by accepting as correct those columns in which only one box has been marked. If no boxes have been marked or if more than one box has been marked the card reader will stop and an alarm will be sounded.

As the trend toward automated preparation of data increases and as more computers are made to talk to other computers there will be more need for error control techniques that are also automatic in their function.

It is then that we shall see more actual use of the error control techniques described in this paper.

It might be said that these techniques were developed perhaps a little before they were actually needed in data systems.

Passing this area under sensing styli can then convert these pencil markings to data signals.

Readout error control is established in the lower or fixed format area of the card by the inclusion of a parity bit level which can serve only for readout protection and be deleted from transmission if desired.

STATE OF THE ART IN SCIENTIFIC COMPUTING

R. W. Hamming
Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

In order to understand the current state of an art it is often necessary to examine its history to see how it evolved. In examining the growth and present state of scientific computing, as contrasted to the other three fields of this session, machine design, software, and business applications, I find that there have been, and are, significant differences—in particular scientific computation has had a much more orderly growth than the others and is probably much more stable now, stable in the sense that the immediate future can be seen reasonably accurately. For example the comparatively ancient text by Whittaker and Robinson can still be used, but a book on coding of five years ago is hopelessly out of date.

Two reasons can be given for the more orderly growth of scientific computing. First, for practical purposes scientific computing has had a much longer history of development. It is true that some of our earliest records are clay tablets recording commercial aspects of a civilization, and it is true that in the 1930's and 40's accounting machines of various types were adapted to scientific work, yet it seems fair to say that scientific computing was far more highly developed over most of the last 2000 years than was commercial computing. Furthermore most of the machines of World War II and the early post-war period were designed by and for men in science.

Second, behind much of scientific computing stands a highly developed and elaborated body of knowledge known as mathematics. This is very true for the field of numerical methods and to a somewhat lesser extent for many areas such as artificial intelligence which I suppose fall in the area of scientific computing. Scientific computing involves the application of engineering judgment to adapt the precise theorems of mathematics to the practical ends of computing, but the existence of mathematical theories is a great asset that none of the other three fields of this session can draw on to any similar extent. The fields of statistics and engineering also contribute to the orderly growth of scientific computing.

In saying that the growth of scientific computing has been, and probably will be, reasonably orderly, I do not want to give the impression of smugness and complacency—there is much that still needs to be done—but I do want to make clear the point that we have a very definite advantage in having a framework of known theorems, accepted notation, and form of publication, against which to judge our field.

I should also observe that both the hardware and the software fields have been generally developed for use in scientific computing before the use in business applications.

Let me now turn to an examination of some special areas of computing, taking them to some extent in the order of the amount of current machine time used. It may come as a surprise to some of the younger members of the computing fraternity to be told that many of the early relay machines as well as the ENIAC and probably the ORDVAC had their motivation in the demands of exterior ballistics—the solution of the ordinary differential equations of

the flight of a missile—in order to produce range tables. Even today many of the simulation problems we find on our computing machines involve the solution of ordinary differential equations of missiles.

A few people may have been aware of the stability problems that can arise in predictor-corrector methods for the solution of ordinary differential equations, but most of us at the end of the Second World War were blissfully ignorant of this fact. Not everything about stability is now known, but we seem to have the problem well under control—except in the so-called "stiff equations" where for the equation

$$y' = \frac{dy}{dx} = f(x, y)$$

we have

$$\frac{\delta f}{\delta y}$$

large and negative. Special cases of stiff equations have been handled, but a good general approach still seems to be lacking.

In the solution of ordinary differential equations it is usual in mathematical circles to speak of the step size—in engineering circles the sampling rate is a better way to describe the situation. In defense of my opinion that matters are well under control I will say that I would be greatly surprised if any new general methods will be found that would allow a significantly lower sampling rate unless they go to using many past data points; the sampling theorem of information theory is against it, more or less. However, for special systems of equations very little is known. For example, there seem to be no generally accepted methods for solving orbit calculations of space missiles in spite of vast sums of money spent for machine time to compare various methods. Nor has the theory so far been able to help much in the matter.

Data reduction is another activity which consumes much time, and which has had no great surprises—except to those who underestimated the amount of data that can be telemetered from a space missile in the course of its active life!

Turning to partial differential equations, we did know about stability in this case and were able to solve on primitive equipment rather difficult cases during the Second World War. The main advance seems to me to have been the discovery of implicit methods of solution which permit us to escape from the net size restrictions. Much work goes on in this area, and I would suspect much remains to be done.

Simultaneous linear algebraic equations have seen an enormous number of published papers, and I suspect the volume is due more to the mathematical elegance of the problem than to the intrinsic importance. In spite of all the research and proposed new methods I keep hearing the words Gauss, Seidel, Jacobi, relaxation—indicating to me that the older methods, somewhat modified are still widely used. Thus I am forced to say that comparatively little progress has been made. I suspect that Householder's modification of Given's modification of Jacobi's method, and Wilkinson's analysis of a slight modification of the Gauss elimination method are the most popular methods at the moment.

Again special cases have found special methods; for example systems with many zeros (so-called "sparse systems") have great importance in many fields of applications. Another example is the special systems which arise in the implicit methods of solving partial differential equations.

Eigenvalues and eigenvectors of matrices have had a similar history; we have had many small advances, we by no means know as much as we wish we knew, but the fields have had no really great improvements, and matters seem to be in a state of orderly evolution.

Zeros of polynomials and more generally zeros of functions are further topics with long histories and no completely satisfactory methods of solution. When I recall that the great mathematician Gauss gave seven different proofs of the fundamental theorem of algebra and all of them are difficult to apply to practical computation, I do not expect to see a really easy method appear tomorrow—but I could be wrong.

Having described some of the more static parts of numerical analysis, let me turn to some of the more changing ones.

I would say that the use of Chebyshev (equal ripple) approximation has produced, and will

produce, great changes in our methods. It now tends to dominate completely the special function approximation area. A few cultivated mathematicians knew about Chebyshev polynomials many years ago, but their importance and central role seem not to have been appreciated before Lanczos rehabilitated them.

Along with the development of Chebyshev approximation I would also place the growth of nonpolynomial approximation in computing. Some methods, such as Prony's method of approximation by sums of exponentials, were known and used, but they remained curiosities rather than regular tools of the trade. In particular I would call your attention to the growth of use of power spectral methods and band limited functions as a significant step forward. Both are as yet in their infancy and much work remains to be done to make their use better understood and more widely appreciated. The band limited function approach goes back at least to Gray and Rubinoff at the University of Pennsylvania, while the name of Tukey is associated with the development of the power spectral approach. Both have their beginnings in electrical engineering practice. Indeed it should be evident that electrical engineering and information theory are two branches of knowledge from which computing can draw new inspiration and new methods.

Monte Carlo methods, meanings using random numbers in a computation, have received a great deal of publicity. However, I would hazard the opinion that the use of random processes, when the original problem does not have such a process, has not so far had the value most of us in the early days expected it would. It is in simulations, which currently occupy a lot of machine time, that random numbers are so often used to simulate various random processes, including noise. A rather large body of knowledge has been developed in this area, and a good textbook would be of tremendous help to those who are not specialists in the area. Indeed, I suspect such a book would stimulate the field itself. Several eminent men have started such books but were too busy to complete them. Thus the field is cursed with such an extensive "oral tradition"—von Neumann knew—it is in Herman Kahn's notes—that the outsider cannot easily get started in the field.

Game theory is closely associated in my mind with Monte Carlo methods. The importance of the ideas of game theory is very great, but so far they have had comparatively little influence on current computing practice. Usually only comparatively trivial situations can be explicitly solved, and this may be the reason for the failure to influence computing practice in other areas such as integration. If so this would seem to be a fruitful field for investigation.

A couple of completely new fields are linear and dynamic programming. Both arose out of attempts at optimization, and are probably not the last fields that will arise from this fruitful source.

I have been surprised at the variety of problems which have been reduced to linear programming problems, especially those requiring integral solutions, but many times it seems to turn out that the general linear programming method is so expensive to use that the reduction is mainly of academic interest. As a result many special cases of the linear programming problem have been investigated and special methods for their solution found. I suspect that much remains to be done and that it will be economically valuable as well as intellectually satisfying.

I am gradually working my way towards such fields as scheduling, critical paths, and mechanization of certain design methods. All of these can, I believe, be summarized under the broad topic of find an "algorithm" for a problem or process that previously had been solved by guess, intuition, experience and dumb luck. This area has a bright future if you judge it by the money involved. Usually there is only a fragmentary mathematical background to draw upon, and this often in the difficult fields of integer solutions, and combinatorial mathematics. The history of mathematics seems to suggest that combinatorial mathematics and number theory are both difficult fields in which to find general methods, and hence we cannot expect to find a supporting structure such as numerical analysis has. The work, therefore, is likely to be fragmentary, irregular in development, and highly frustrating at times. Yet, let me repeat, it probably has a bright future in the sense that spectacular, unexpected results can be found in many special, important cases.

I would like to summarize a couple of points I have been developing. Optimization, and discrete, integer processes are two threads of scientific computing which have recently given rise to much new material, and should produce even more in the future. Both lack an adequate, practical mathematical theory for background, but both are of great economic importance, and support for work in these areas should be easy to find in the form of special situations of urgent importance to management.

I have avoided to a great extent discussing topics which have statistics as opposed to mathematics as the formal background, as well as the topic of statistics itself. Clearly statistics is required in the analyses of roundoff effects in long computations, in the design of experiment plan for examining situations involving many parameters, in sampling plans, in significance tests connected with "least squares" fitting, etc. Here again we have many topics awaiting further development, but for which much of the formal background information is reasonably well developed. In particular, in Roundoff Theory our knowledge is still in a sad state in spite of its obvious importance and extensive bibliography.

Statistics itself seems to be changing somewhat due to the impact of computers and the problems that are arising from the available data that can now be processed economically.

Topics in the general area known as "artificial intelligence" seem to fall in the domain of scientific computing, but unlike most of the topics so far discussed there is seldom a body of precise knowledge behind them. Most of them are in spirit like the other three areas in this session, subject to uneven growth, surprises, our inability to judge easily the significant result from the trivial and, even at times, false results.

I think this is the proper time for me to develop the themes of the advantages and disadvantages of having a body of precise knowledge like mathematics behind a field of activity. In trying to do this let me contrast research in numerical analysis to research in artificial intelligence. The vastness of the known relevant material in mathematics means that one cannot just sit down and write a paper with some new results in numerical analysis; rather a long apprenticeship is necessary. On the other hand, the ideas are available, the notation and style of presentation are well developed (one merely writes up the mathematical aspects and almost totally ignores the art and engineering judgment involved), so that it is in this sense easy to do research in numerical analysis. Just the opposite is true in artificial intelligence; almost anyone can sit down and imagine some new attack, but it is difficult to carry it out and especially difficult to present it in a written form suitable for publication. Furthermore, in the artificial intelligence area startlingly new results are of frequent occurrence and hence stimulating to further work. However, it is difficult to judge the significance of one's work and to know whether it is along a "good" or a "bad" path. In short there are advantages and disadvantages to having a well described body of knowledge as a background for activity.

I was asked to comment on what we did in the past that was wrong and what was right.

I believe one of the errors in numerical analysis is that we have too long tended to regard it as a branch of mathematics and to believe that the mathematical theorems were more relevant than we should have. Thus a corollary of the fundamental theorem of algebra states that $1,x,x^2, \ldots, x^n$ are linearly independent in any interval, but the Chebyshev polynomials show their dependence in the presence of noise; $T_{21}(x)/2^{20} = x^{21} + \ldots$ and is less than 10^{-6} in $-1 \leq x \leq 1$. Of the right moves, we have been aggressive in opening new areas of thought like linear programming, Monte Carlo methods, etc. and if at times we have overestimated their importance that was far better than to have underestimated them. Another error we have made is to fail to produce textbooks in various areas such as Monte Carlo methods and hence the newcomer or outsider has difficulty in getting started easily.

I suppose I am expected at this point to take a grand view of scientific computing and briefly summarize years of development by thousands of workers in the world. It is an impossible task to do justice to, but here goes.

The heart of scientific computing, namely numerical analysis has had a fairly steady, reasonably controlled growth in the past decade. Three main classes of problems, often overlapping, namely, simulation, optimization, and combinatorial-integer problems, have given rise

to new disciplines and will probably spawn even more in the future. At present there are dozens of areas of specialization; in the near future there will probably be hundreds. The process of finding algorithms for areas previously considered as requiring thinking has also produced whole new fields such as theorem proving, drafting, language translation; more will arise.

The problems of becoming acquainted with what is known is now hard, and it will only become harder in spite of all we manage to do with machines to help out. While the need for good, clear, simple presentations of known and new results is increasingly great, very few people seem willing to do much about it.

A CRITICAL REVIEW OF THE STATE OF THE PROGRAMMING ART

R. S. Barton
Altadena, California

. . . When once the engine shall have been constructed, the difficulty shall have been reduced to the making out of cards; but as these are merely the translation of algebraical formulae, it will, by means of some simple notations, be easy to consign the execution of them to a workman. Thus, the whole intellectual labour will be limited to the preparation of the formulae, which must be adapted for calculation by the engine.

L. F. Menabrea, 1842, from Sketch of the Analytical Engine Invented by Charles Babbage

INTRODUCTION

What follows is intended not as a scholarly review of the programming art, but as a personal appraisal. The bias is that of one interested in the subject of machine organization and its relation to programming.

Surveying the still small body of computer literature, one is struck by the scarcity of substantial material on programming as such, and might wonder if the subject of programming is inherently trivial, or whether its study is being neglected as a separate discipline.

The writer thinks that while programming has developed as a practical art rather than as a science, present trends suggest that a formal theory of programming will emerge and become an important branch of mathematics. It is not clear why those with formal training in mathematics and logic were not attracted in larger numbers to the subject in the preceding two decades; and it is now difficult to predict how quickly a formal theory of programming will affect existing technology in the computer sciences.

THE PRACTICAL ASPECTS OF PROGRAMMING

Superficially, programming is an easy skill to acquire. A few hours of instruction on a particular machine or language is enough for a start. Aptitude varies and successful programmers, in the past, acquired much lore which became part of their stock in trade. Today, however, many of the machine peculiarities and operational technicalities are handled in the workings of system programs so that the practical difficulties of programming have been alleviated for the average machine user. The professional programmer designs and constructs these programming systems.

The practical aspects of programming today consist of (1) the design of suitable languages to express problems of interest, (2) the instruction of people in the use of these languages, (3) the maintenance of a library of programs to economize on effort, (4) education of users in techniques of efficient program organization, and (5) the designing of the

translators and systems for the machines. All other considerations are extraneous.

Language Design

A programming language must meet, ideally, these criteria: (1) it must enable precise description of any representative of the class of problems for which it was intended; (2) it must be mechanically translatable to a common standard; (3) it should be concise; and (4) it should take into account the preferences of the average user.

The user's preferences are influenced by his training and experience. Within a field of application, programmers are likely to have similar educational backgrounds, and hence, familiarity with certain notations and language forms. Then, too, the nature of the human mind, eye, and nervous systems may predispose people towards certain notational forms.

Machine characteristics should no longer be allowed to influence language design in any way. Matching the human language to the machine language is the task of the system program. Compromises tending to favor machine requirements have been often detrimental, and, as an intermediate measure, hand transcription—as a separate clerical task—might be preferable to the use of a restrictive notation in the case of certain problem-oriented languages.

Teaching the Language

A language is characterized by its vocabulary and a grammar, and recently, beginning with the description of ALGOL in Backus notation, precise and concise descriptions of syntax have appeared. The value of such a precise description is not yet fully appreciated as an aid to learning and subsequent use.

Instructional texts have been prepared both in linear and branching programmed text form. Practice in the writing of programs can be made more convenient in some cases through use of an automated programming laboratory.

These developments in methods of precise language description, techniques of presentation, and economical methods for providing machine practice are all potentially very important in teaching fluent and widespread use of programming language. It is desirable to teach both reading and writing of programs rather than, as in the past, only program writing. As an aid to encouraging correct grammar, the existence of a precise syntax for a language permits mechanical checking of correct usage as an additional feature of automated instruction.

It would now seem both feasible and desirable to delegate all programming language instruction to a computer program provided with each computer.

The Library Problem

The importance of a library to provide a repository for programs was recognized early, but full exploitation has been impeded by poor program documentation, lack of interest on the part of programmers, and language problems. Many program libraries now consist of programs in languages either dead or destined for an early demise. Limitations result from the dearth of program "readers" and the serious practical difficulties in translation between machine languages.

Libraries will, of course, be stratified, ranging from the personal through the semi-private to the public; and classification and abstracting systems become increasingly important as programs enter the public domain. Part of the education of programmers must be directed to the construction of the most general routines likely to be of use, since such routines rather than specialized ones, are most likely to be useful items in a library.

A standard language is needed to ensure permanence and maximum utility for programs in the public domain. This need not be directly readable, and hence, can be a universal machine language.

Automatic documentation techniques of flow diagram synthesis and language expansion are relevant to library maintenance and convenience of library usage. Library space is less of a problem than in libraries containing books. Programs can be in hierarchical form, it no longer being necessary to accept the constraints of serially accessible storage to allow efficient assembly.

It is important to note that in addition to explicitly referencing the library in the program, an ever-increasing amount of implicit referencing will result from the use of problem-oriented (as distinguished from procedural) languages. Thus, the geometric languages cause the selection of library programs

to carry out geometric solutions implied rather than stated procedurally. Use of the pertinent part of the classification system is built into the translator.

Teaching Programmers the Techniques of Program Organization

Program organization is, in some respects, the heart of programming, and should be independent of the processor characteristics; though, in the past, these have intruded prematurely in most analyses. Perlis has summed up the matter in seven words: definition, sequencing, selection, substitution, binding, replication, and evaluation. It is unfortunate that, as yet, there is no sufficiently general standard notation to represent these programming concepts. Fragments appear in various languages; examples are the block, conditional, and for statement of ALGOL. The flow diagram notation is the most common method of representing structure, but is a cumbersome device for use with complex programs unless a hierarchy of diagrams is used.

Iverson, in his recent book, "A Programming Language,"[1] defines general operations on structured operands such as vectors, matrices, and trees at one extreme and descends to a level of detail pertinent to the logic designer at the other.

The writer believes that the representation of structure is the most important aspect of programming for purposes of formalization. With this accomplished, suitable basic education would be feasible in the schools, and a mathematics of program transformations might follow. An eventual consequence would be the mechanical resolution of the problem of determining the best arrangement of a program subject to given constraints.

Designing Translators and Systems for the Machines

The systems programmer now has the responsibility of statisfying the user-programmer and making-do with a machine that is already in existence, or, equivalently, completely specified. The second requirement which now presents serious difficulties need not be a permanent obstacle, since the system programmer will enjoy increased participation in the design of future machines. Of course, this poses the problem of educating system programmers in the subject of computer organization.

A machine language need not be suited for direct use by human beings. This is convenient only if one allows for a translation in both directions between person and machine. If the fact is accepted that a machine language need never be seen *or directly manipulated by* programmers, some artificial constraints in the language design are removed, and its representation in the machine insofar as both format and codes are concerned is a problem that need concern only the logic designer.

For the immediate future machine language will be of procedural type, whereas the convenience of nonprocedural languages for the programmer is widely recognized. The syntax of the machine language seems likely to be simpler than that of the usual programming language. Most important, the essential part of the machine language will be descriptive of program and data *structure* and the choice of evaluative operations is of secondary importance, since in principle as few as one logical operation is sufficient to synthesize higher level operations, provided the structure of the programs can be represented. Putting this last point in a somewhat different way: The evaluation-type operations required can always be effected by subroutines, and these can be made as efficient as desired by use of high-speed storage, concurrent operation, and compact coding.

HISTORY AND THE PERIOD OF COMMERCIALISM

The Pioneers

The recently published book, "Charles Babbage and his Calculating Engines,"[2] makes fascinating reading and helps to provide some perspective. The book contains notes by Ada Augusta, Countess of Lovelace, who could be justly described as the first programmer.

After the work of Aiken and Stibitz independently led to the first automatic computers in the early 1940's, the first programs comparable to those conceived for the Babbage Analytical Engine were prepared and executed, though prior to this, astronomers had used punched card machines in a manner analogous

[1] Wiley, 1962.

[2] Dover, 1961.

to Babbage's proposed Difference Engine. It is interesting to note that Aiken, realizing that the preparation of computer codes was a demanding and fatiguing task, designed a coding machine with keyboard labelled with appropriately arranged mathematical symbols. Aiken also foresaw the coming shortage of adequately trained personnel, developed some of the earliest academic courses, and taught many of today's leading theorists in the computer field.

In the mid-forties, when the stored-program computer had been conceived, Goldstine and von Neumann prepared a report which was fairly widely distributed, but not formally published, entitled "Planning and Coding of Problems for an Electronic Computing Instrument." This report contains one of the earliest discussions of program structure and its representation by flow diagrams and is worth reading today. During this same period, Mauchly developed coding methods for the new stored-program machines.

The Commercial Era

The present period began when Univac, which was to be the first commercially available large computer, became, as a result of an airplane crash, a "business machine," and its inventors, Eckert and Mauchly, were no longer in control of the manner of its introduction to the market. The business machine industry, managed by executives of the pre-electronic era, absorbed the few scientist/engineer-run computer firms, and imposed on a new, struggling technology the ideas and practices of the punched card and office machine industry. The promotional and sales effort that followed has had, in little more than a decade, the effect of making computers a familiar part of our industrial, business, and government operations, but the competition has had some unhealthy side effects pertinent to the development of programming as a profession.

The intense competition of the early fifties quickly turned instead into a quasi-monopolistic situation dominated by IBM, with a semblance of competition maintained through fear of anti-trust action, some equity in distribution of government orders, and the rapid growth of the market. The programmer, during this period, was in the paradoxical position of being much in demand and the recipient of ever-larger paychecks and yet was often regarded as being of subprofessional status. This followed, in part, from the efforts of the marketing man, who, with his semi-technical sales-support and advertising staff, has promulgated so much nonsense about computers that the programmer, in the position of having to make good the claims, has been often only partially successful, and sometimes foredoomed to failure.

During this time, the universities were relatively inactive in computer design and programming theory. Since the manufacturers allocated funds mainly to the development of new components and hastily-conceived designs for the market, technical progress in machine organization and programming was erratic. The relation between the ever-growing programming problem and machine organization was overlooked. The programming problem, finally reaching massive proportions, was attacked via programming systems, also hastily designed.

The professional general-purpose applications programmer was soon obsoleted by an abundance of cheap machine time and the delegation of nuisance aspects of programming and machine operation to the inside of system programs. Somewhat amusingly, the machines became their own best customers with FORTRAN, a leading contender for machine time, not because of quality of compiled program, but rather because of its own lengthy compilation process.

There was a hasty exodus of the more skillful into the new field of system programming, a specialty in which it was quite possible to forget the purpose of computers in the face of challenging problems to solve.

Emphasis on the old goal of efficient machine utilization diminished and attention was refocused on systems. User organizations, dedicated to sharing and good fellowship, the salesmen selling "software," and the language standardization committees were typical of the new period.

The user organizations, pioneered to share developmental cost and programmer talent in support of new machines, soon degenerated into pressure groups intended to persuade the manufacturers to provide system programs and other support for their products.

The former self-reliance of many of the major computer users was replaced by dependence on the manufacturer, and many of the

most capable programmers from the user installations gravitated to the manufacturer's support activities.

One may wonder where the foolish term, "software," originated, but need not doubt that it was eagerly seized upon by the gimmick-hungry salesmen. There is something conveniently intangible about "software," and an agile, alert salesman can make promises as his needs dictate. Irresponsible marketing and unreasonable customers combined to furnish continual harassment of the programmers supposed to produce the systems, and who were already handicapped by poorly defined machines and languages and the lack of a sound technical basis for the work being attempted. Subject to whimsical sales pressures and working, as is often the case, directly under nontechnical marketing management, the results were seldom satisfactory, often near-fraudulent.

The language standardization issue, by committee or de facto, had three main episodes: FORTRAN, ALGOL, and COBOL.

FORTRAN was sufficiently useful to achieve widespread use and was known to be a program of great size and sparsely documented complexity. It was the first compiler to be widely advertised and aggressively sold to the market, and not surprisingly, a user-cult formed which is now quite effectively hampering progress in the adoption of improved scientific languages. For some time, superior methods for implementing a language of the comparative simplicity of FORTRAN have existed. Useful translators have demonstrated this fact. But still the usual FORTRAN compiler is little improved over the earlier systems and the few exceptions to the rule are not widely used. The user remains satisfied with a limited language and absurdly inefficient translation. The user excused indifference to improvement by pointing out the investment in program libraries, the difficulties of conversion, and the cost of retraining.

ALGOL was an attempt to make a step beyond FORTRAN and achieve, at the same time, a standard. Manufacturers' representatives, members of the academic computer world, and prominent computer theorists from abroad made contributions. Programmers in this country seemed uninterested in ALGOL, in contrast to the Europeans and Soviets, criticizing it in the superficial way that has its origins in ignorance. This opportunity to introduce some professional standards in place of crass commercialism was rejected because of such factors as mental laziness in the face of the Backus notation, awe of recursion, and the use of a larger character set than was then available for existing equipment.

COBOL was a rather different phenomenon. Bureaucratically inspired, industry politics ridden, and representative of an eccentric view of programming, it and its forerunners seem in retrospect to have been predicated on the notion that inability to learn concise notations could be circumvented by providing a kind of pidgin English. The possibility was overlooked of making programs intelligible to the nonspecialist by language expansion during translation. It is unfortunate that the real contribution of the business application programmers, the notion of data descriptions logically separate from the procedural description, was obscured by association with narrative-style language.

Machine Organization and the Programming Problem

It is true that the earliest machines were, from a programming standpoint, often logically more elegant than their successors. Anxiety about circuit reliability was a motivating factor behind their simplicity, but it should also be remembered that the logical organization was usually the concept of one, or at most a few, closely associated inventors.

By the time that computers had become more reliable, the commercial organizations had taken over and the committees and other foibles of human organization did violence to the logic as the demands for greater speed and ease of use increased and more elaborate computers were built. The clumsy designs which followed inspired application of the word "kludge" and programmers, guilty more by omission than commission, were wryly amused.

It is notable that even as early as 1947, Stibitz, at the Harvard Symposium on Large-Scale Digital Calculating Machinery, pointed out the intimate relation between machine organization and the practical usefulness of the machine in terms entirely appropriate in 1963.

A difficulty is that both designers and programmers are now overspecialized. A programmer to be professional must not ignore

the subject of machine organization, since programming and machine organization are inextricably intertwined. The machine designer, on the other hand, cannot remain ignorant of programming in the general sense; he must, for example, have some knowledge of the problems of translation of mechanical languages and an appreciation of the control problems arising in the preparation and use of very large programs.

The designer is still under marketing influence, defined here as what the salesman thinks the customer wants or what he has convinced him that he wants, and speed measured in terms of time for individual memory accesses, arithmetic operations, or simple loops provide easy, if inadequate measures. Marketing, of course, wants the latest glamour component since the user has learned, with some reason, to equate the new devices with easily observed improvements in performance. The result of these pressures is a neglect of improvements in machine organization, improvements which could enhance over-all performance of systems and remove an obstacle to substantial advances in programming system design.

Programming and Education

The computer sciences are beginning to find a place in the university. Still overshadowed by matters of computer procurement and demands for internal services, there is now considerable serious research going on in programming and machine organization, automata theory, switching theory, and mathematical linguistics.

Programming is not trivial, and to quote from the previously cited report by Goldstine and von Neumann: "Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics." For some reason, formal treatments of programming have been both specialized and rare. The idea that the construction and description of algorithms is a legitimate concern of mathematical instruction has its advocates. In view of the major revision now taking place in the mathematics curriculum, it would seem timely to put programming on a formal basis for mass education. Neither existing machine languages nor the one-level-removed programming languages will do for this purpose. Considering the enormous impact that the computer is certain to have in education, the necessity of avoiding excessive influence from the computer manufacturer must be recognized by all concerned.

THE DESIGN AND CONSTRUCTION OF PROGRAMMING SYSTEMS

Some of the important ideas that have filtered down into current practice are reviewed in the following sections in roughly the order of occurrence. Since the development of programming techniques and concepts is poorly documented, it is difficult to assign credit for origination of the ideas or their succesful application. Workers in the field know that most of the useful concepts had multiple origin. Significant ideas have been embodied in useful programs, but not otherwise documented, long before publication. In view of the scarcity of program readers and the lack of standards in language for description, one might say that the work in point was not "published," though its existence is implied by the operating programs.

Combining Programs, Assembly

The importance of combining programs from different sources was recognized early and a program to accomplish this was described by Goldstine and von Neumann.

Today, the assembly process is often a subordinate function in a compiler and symbolic forms of machine language are rapidly declining in use in the technical application area.

Interpretation

Interpretive programs were important for a number of years in programming technical applications, and it is interesting that one of the first, developed at MIT, accepted an algebraic language. Most of the later programs simulated multiple-address machines with floating-point arithmetic, index registers, and built-in mathematical functions. Interpreters were successful in spite of their inefficiency because many of the deficiencies of the machine could be easily corrected in the simulation and, thus, in addition to more convenient and compact codes, checkout was greatly simplified. The success of these interpreters helped to ensure the incorporation of more powerful operations

in the popular machines of the middle fifties which change, together with the development of compilers, caused the interpreter to pass out of use.

Compilation

The most promising solution to the programming problem was clearly compilation and, because of work volume, was conceived first for business data processing use. The first attempts were not outstandingly successful from the use standpoint. Excessive compilation time, difficulties in checkout that encouraged patching in machine language, and relatively bulky generated codes tended to discourage users.

The first languages were compromises of programmer convenience, complication requirements, and machine characteristics. Now that the compilation process is better understood, and machine designs less restricted, the programmer should soon have the use of the more powerful languages now practical.

The deficiencies of some of the first compilers stemmed from neglect of integrated design, use of existing programs for the assembly process, and storage limitations in the machines.

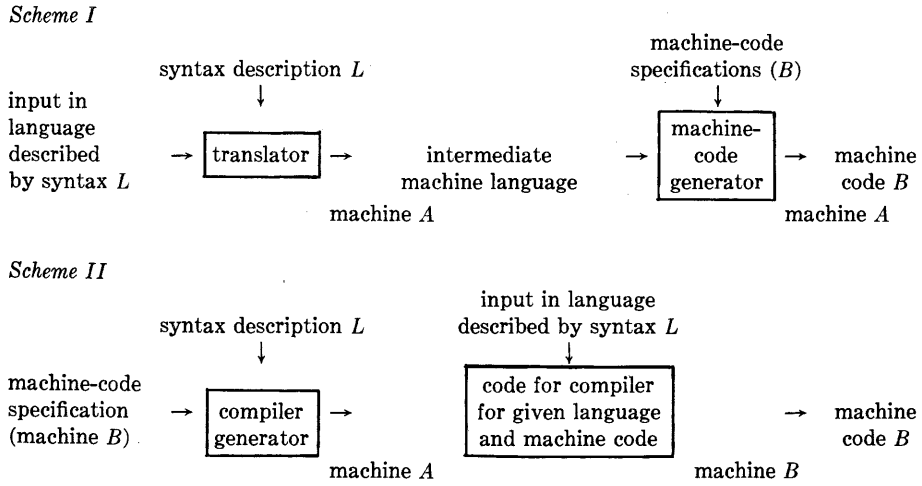
In the scientific systems, translation and code generation were conceptually separated into two problems through use of an intermediate language describable as that of an "ideal" object machine for the translator writer. The use of an intermediate language (often three-address or Polish) was one of the first important advances in compiler design.

Many improvements of technique were discovered only in recent years: the incorporation of a simplified assembly process as an integral part of the compilation, the use of linked lists for storage pools, stacks for analyzing tree structures in the input language, improved symbol dictionary techniques to eliminate bulky sorting passes unnecessary for the symbol-handling requirements of a compiler, and the coupling of sequential processes via buffer areas in storage to minimize intermediate storage on magnetic tape are notable examples. These techniques have made possible compact, fast, and inexpensively constructed compilers.

Later ideas applied successfully include the organization of a translator using recursive subroutines to handle control; and the writing of a system in its own language, followed by hand translation of a sufficient part of the code to allow compilation of the entire compiler program.

Syntax-Directed Compilers

Syntax-directed compilers are representative of the latest thinking in the field. In one plan (see the diagram) a translator is constructed for a machine A which for some class of languages described will produce as output a machine-independent intermediate language from input in language L. Statements in the intermediate language are then used as input to a machine-code generator, which produces actual machine code for machine B. Then, if translator and generator sections for use with ma-



Syntax-Directed Compilers

chine B are written in language L, the system can be transferred to machine B. The machine-code generator might itself be of a general nature in that specifications for the object machine B might determine its output. In another plan, a compiler-generator on Machine A produces the code for a compiler to run on machine B given the syntax of the language L and the machine-code specifications for the object machine B. In practice, the intermediate language has only a transitory existence within the machine.

Just how close these schemes are to practical application in their entirety is not known as published material is scanty: possibly the techniques are regarded as proprietary. There has been much speculation about and occasional partial demonstration of these ideas during the past few years. Both notions need clarification.

Theoretical investigations by Chomsky and others on formal languages are beginning to influence practical work. The least clear aspects remain in the areas of machine-code specification, and the writer suspects that the general machine-code generator of Scheme I is ill-defined in a practical sense, and perhaps presents the same order of difficulty as a machine-language-to-machine-language translator. The first part of Scheme II presents a similar difficulty, but by properly associating the syntactical elements of the language with desired machine code, a relatively simple program might serve for the first step, although it is not obvious what restrictions would have to be placed on the syntax and machine code.

A Standard Machine Language

A simple, and the writer believes effective, solution to the problem is in standardization on the transitional language of Scheme I as a symbolic *machine language*. The machine-code generator is then a straightforward program which handles differences of representation and makes substitutions for evaluative operations not available in the object machine. The description of the object machine relative to the standard language would consist of code representations for standard operators, code bodies in the machine language equivalent to omitted evaluative operators in the standard language, and *with limitations,* operator symbols in the machine language *not* corresponding to evaluative operations in the standard language, together with their equivalent symbol strings in the standard language. In the foregoing, the term, "evaluative operator," is used to distinguish from control and descriptive operators since the standard machine language would be complete and definitive in that regard.

Storage Allocation

Storage allocation was the primary function of the early assembly programs and continues to be an important part of a modern programming system. A resident program is given the responsibility of handling the grosser storage allocation problems because many machine environments require this to be done dynamically.

Since multi-level storage is an economic necessity, its efficient utilization has long been a prime objective, though the difficulties appear formidable. In this regard, the machine designers have largely ignored the problem; extensive programmed control, very costly in terms of storage, is usually the consequence. It now appears that things can be done to alleviate this problem by providing adequate interrupt signals indicating states of tape, disc, and drum storage components and, through use of simple adaptive schemes for handling information transfers between storage levels.

The incorporation in procedural programming languages of notations for describing data structures such as arrays, files, and trees, and the provision to use these structures recursively together with indications of the scope of definition, will help greatly with the storage allocation problem and assist the programmer organizationally, and yet not burden him with need to cater to special machine characteristics. The structuring of programs should be, of course, an important part of the education of machine users.

Operating Systems

If operating systems are defined as a means of automatically accomplishing gross storage allocation, the ordering of programs, and the assigning of processor subsystems, it is reasonable to expect that these functions be mainly built into the equipment as a judicious combination of circuit logic and fixed program. That bulk of present-day systems is an implicit criticism of the machine designs. In the allocation of tape control, processor, storage, or input-

output subsystems, experience shows that if a choice must be made between equivalent units, the hardware should make the assignment. Automatic interrupt logic should make available error signals and state indications for the units involved, and should cause the automatic preservation and restoration of machine states, and thus, is closely related to mechanisms for the handling of subroutines.

Levels of Language

If machine languages are thought of as first-level languages, then procedural languages are second-level and problem-oriented languages third-level. ALGOL is an example of a second level, and the geometric language, APT, the third level.

Often a problem-oriented language will contain a procedural-type language as a sub-language, and ideally, that procedural language should be that in which the system is written and extended.

A procedural language should not contain a machine language as a subset, since this, in effect, is tantamount to admitting that the procedural language is incomplete, or that the machine is of such peculiar characteristics that rules for its use cannot be subordinated algorithmically within the translator for the procedural language. The fact is, however, that procedural languages, as defined at present, tend to be incomplete, and this is presumably compensated for in a nonstandard fashion by augmenting the language for each compiler, or including machine language. Machine pathologies are still with us, and this will be the case until the design philosophy in fashion is replaced and the machines now in use are obsolete.

A wealth of problem-oriented languages must be expected, since these will be the most powerful means of communication with machines, and their translators will manage the program libraries which grow up about the specialties for which the languages are conceived.

Variations of taste will ensure a number of procedural languages, but these are likely to be very similar. Standardization at level two might be desirable, but the difficulties seem tremendous and the cost of not standardizing will not turn out to be forbidding if some rationale is developed for level-one languages.

The reasons for rigid standardization at level one have been mentioned elsewhere in this paper, but are worth repeating. Machine languages need not take account of the questions of taste or humanly suitable representations, since virtually instantaneous translations *in both directions* between person and machine are clearly practical. All programs are expressed in terms of the same structural elements, and thus the designer has a standard which is completely independent of the proposed class of applications for the machine. In a given design, some characteristics could be emphasized at the expense of others (though probably not in a true general-purpose machine), but all control and data accessing functions would be present. The particular representation of elements in the machine language employed is also at the option of the designer, but each element must be available separately and be combinable in all meaningful ways.

In the sense just described, there can be a standard machine language acceptable to both machine designers and system programmers, and this language would be the logical and economically most satisfactory level for standardization. By developing a standard for machine language, both programming and machine design can proceed in a more orderly fashion with better use of valuable technical manpower.

COMPUTER APPLICATIONS FOR INDUSTRY AND THE MILITARY—A CRITICAL REVIEW OF THE LAST TEN YEARS

Donald F. Blumberg
Director, Planning and Information Systems
Pennsylvania Research Associates
Phia., Penna., and member of the Staff
Moore School of Electrical Engineering
University of Pennsylvania

I. INTRODUCTION

The utilization of data-processing equipment in the United States over the last ten years has been growing at a phenomenal rate. Digital computing systems are now in use in almost every sector of the economy, and employed in an extremely wide variety of applications. The data processing field has come a long way from the early stages of computer development when systems applications could be divided into the extremely simple classifications of "business" and "scientific" use. When discussing the applications of data-processing equipment in the early 1950's, one had in mind either their utilization for accounting and standard payroll functions, or the processing of complex mathematical formulae which would be too unwieldy and complicated to do by hand. There were a few individuals on the far-out fringe of the field who suggested that electronic data-processing equipment would some day have a profound effect on wholesaling, retailing, the control of complex processes, and the entire middle management field. But, by and large, most people were primarily intrigued by the complexity of the technology rather than the specifics of new application.

When the *Univac* I was spanking new, back in 1953, those few individuals who were premany as 3,000 computers within the U.S. economy were simply ignored as being wildly optimistic. In ten short years we have progressed to the point where over 10,000 data-processing machines are now installed, and computer applications run the gamut from allocation algorithms to weather processing. This ten-year range is a fairly long period of time, and is sufficient to begin to draw some conclusions dicting that there were applications for as about the applications development process and identify those functional areas for which computers have been successfully applied. Typical application surveys are often not very helpful in that they simply enumerate as extensive a list as possible of the range of applications with as many references as space permits. However, in the long run it is not very significant that Company X is now using a computer to maintain an accurate warehouse inventory. The key issue is the degree to which a particular class of industry is utilizing computers. We can assume that there is a standard range of functions, by industry, that computers can perform. In other words any important attempt to structure, and delineate the applications of computers in the United States must first quantitatively analyze the range of installations by sector of the economy. This critique will also involve an examination of the pathways used in developing and applying the computer to various functional problems in the economy.

and to place needed focus and emphasis on some key underlying factors in the development of new applications.

There are some who might question why this historical review is required. The computer industry has always been looking forward; why not just make the pat, optimistic predications about future applications and be done with it? The fact is that the biggest single problem in the computer industry has been the lack of competent and realistic planning. The absence of a reasonably extensive performance pattern upon which to base future predictions, and to structure future developments has often led to a situation in which a new technical achievement was enough to generate an entirely new range of apparently "believable" speculation. It is essential to use the past as a structured method of at least delineating the future. By objectively criticizing past developments, we can begin to understand the real underlying factors in achieving success or failure in this industry.

In trying to summarize this extremely complex situation in a relatively short paper, the writer has been forced to present the facts in an abridged and somewhat biased form, painting the blacks a bit darker, and the whites a bit lighter, in order to emphasize certain general conclusions. Critical license has been taken, and the reader should be, and is herewith forewarned.

In summation, the general intent of this paper is to quantitatively examine the range of computer applications in the U.S. economy and to qualitatively consider and criticize the efficiency and effectiveness of the development process which led to the present situation.

II. GENERAL TRENDS IN COMPUTER APPLICATIONS

The United States computer industry has been growing at an extremely rapid rate since the introduction of commercially available equipment in the latter half of 1952. Some idea of the extent of this growth can be obtained by looking at the market for data-processing equipment and services during the period 1953-63 (Exhibit 1).

Two historical trends in this expanding field deserve some comment: 1) the inter-relationship between the special military systems and

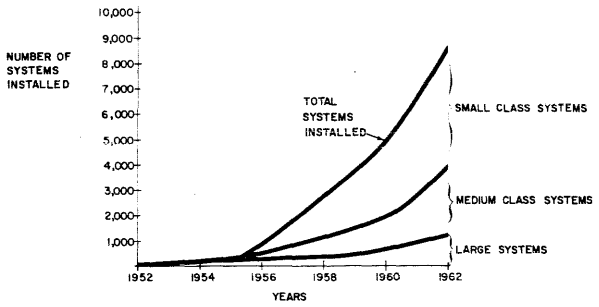


Exhibit 1. Number of Digital Computers Installed in United States, 1952-1962, Small, Medium, Large.

standard computer systems market, and 2) the growing market for data processing services. The military system market has always been a keystone of the data processing industry. For example, military expenditures for research and development of advanced computer hardware and "software" has given a tremendous degree of technical impetus to the field through the years. Military requirements have also led to the implementation of a wide range of new applications for information processing, weapons systems control, command control, etc. Although most of these applications originally utilized specially developed equipment, the requirements are now being satisfied by commercially available units. In fact, as will be indicated elsewhere, the National Security Sector of the economy is a major user of commercial data processing equipment. The military requirement for complex weapons and supporting equipment has also led to growth in the utilization of computers in other sectors of the economy. For example, the large aero-space and electronics manufacturing concerns working on major government contracts have also widely applied computing equipment in direct or overhead support of these government programs. Thus, both directly and indirectly, the federal government's expenditures for goods and services formed the basis for a large number of data processing applications during the last ten years.

A second major trend which has occurred in the last decade has been the rapid rise in the development of computer services. The major problem in the data processing industry, at first, was simply the design, development, and production of reliable, efficient processors. It was believed that the user would easily develop

and program his own applications once the machine was made available. In the latter half of the last decade, with the appearance of a wide range of equipment, it became quite clear that the development and programming of the application, and even the operation of the processing system could not be ignored. The result was the development of a whole new market for computer support services including:

- Applications analysis
- Development of computer programs
- Operation of computer system, providing available processing as needed (e.g., the EDP Service Bureau)

This new field has grown to the point where the availability of these services can exert influence on the development of new applications. For example, in the banking field there were very few people who had the technical skill and competence to analyze operations and develop working programs. The introduction of consulting firms and programming services with bank systems skills furthered the development of this particular application. The availability of data processing time through service centers on an as-needed basis, without major investment in a machine installation, has led many new firms to begin to apply data processing techniques.

A better understanding of the nature of computer applications in the United States during the last ten years can be gained from an analysis of the trends in the number of installations of small, medium, and large systems (Exhibit 2).

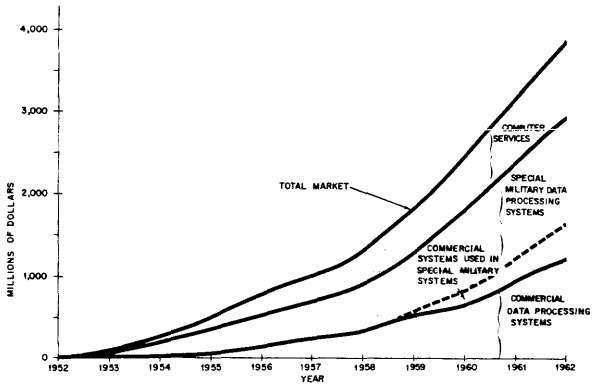


Exhibit 2. Market for Computer Systems and Services in the United States—1952-1962.

The exponential growth in number of units installed since the 1956-1957 period has been in the small to medium class categories. Large-scale systems installations have also increased, but at a much more stable rate. Application of small systems to date have been largely for accounting and administrative operations. A significant percentage of these applications have been upgraded replacements* for punched card and tabulation installations.

The ability of the economy to accept so many of these small to medium size units in such a short period of time becomes less surprising when the replacement units are discounted. However, there are a number of computer applications for the small to medium size user which have been developed in the last few years to provide the basis for these new installations.

A final important factor in the trend of computer applications over the last decade involves the critical inter-relationship between the development of an applications concept, and the availability of an *appropriate* computer system upon which to implement the concept. A correlation of the availability of various types of computer systems (Exhibit 3) with the rate of installation and development of various classes of applications would clearly demonstrate the importance of various systems configurations. For example, the availability of the IBM 305 RAMAC System opened up a wide range of inventory management applications which had been impractical with previously available systems. On the other hand, the lack of availability of an economical, yet efficient *system* for information storage and retrieval in a price range which most users consider to be acceptable has been holding back this interesting application.

The extensive use of a particular application is closely tied to the availability of an efficient system for processing the application.

---

* The replacement factor in the growth of the computer market has been apparently neglected by almost every manufacturer except IBM. Replacement of unit record equipment with computers, and the replacement or augmentation of one computer with another has been a major element in the apparently high installation rate over the last three years. This installation upgrading has almost always been accompanied by an increase in the variety, as well as the complexity, of applications on the system.

1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962
UNIVAC I	IBM 701	BURROUGHS 205	MONROBOT III	IBM 704	IBM 705 III	IBM 610	NCR 304	GE 225	RCA 501	BURROUGHS B270-280
		ALWAC IIIE	IBM 650	IBM 705 I,II	IBM 305	BURROUGHS 220	IBM 7090	IBM 7070	RCA 301	ASI 210
		IBM 702	BURROUGHS 101	UNIVAC 1103A	UNIVAC II	PHILCO S 2000	UNIVAC SS80	IBM 1620	NCR 310	NCR 315
		UNIVAC 1101	BENDIX G-15		DATAMATIC 1000	UNIVAC 1105	UNIVAC SS 90	CDC 1604	BENDIX G-20	IBM 7074
			LGP 30		RECOMP I	IBM 709	RCA 501	CDC 160	RPC 9000	
			UNDERWOOD 100		UNIVAC FILE COMPUTER	RECOMP II	PHILCO S1000	RPC 4000	CDC 160A	
								HONEYWELL 800	IBM 7080	
								LARC	UNIVAC III	
								PHILCO 2000-211	HONEYWELL 290	
								IBM 1401	HONEYWELL 400	
								PB 250	RECOMP III	
									UNIVAC 490	
									IBM 1410	
									BURROUGHS B250	
									NCR 390	
									STRETCH	

Exhibit 3. Date of Initial Installation of Computer Systems.

III. A SURVEY OF COMPUTER APPLICATIONS IN THE UNITED STATES

Although computers are to be found in almost every sector of the economy, their applications in specific sectors vary, depending on the nature of the problems to be found, the array of technological equipments available at a particular point in time, and the existence of application concepts which have been proved feasible. An analysis of the degree of use of small-medium class, and large class computer systems in various sectors* of the economy (based upon a percentage of total installations) is depicted in Exhibit 4.

In general, the Manufacturing sector and the Federal Government sector of the U.S. economy are the largest single users of the ten major sectors of the U.S. economy. However, there is a difference in the degree of utilization of small-medium versus large class systems in these different sectors. These differences are caused by the variations in computer applications functions in each of the sectors, as well as the size and structure of each of the industrial segments. For example, certain segments of the economy are so fractionalized that very few firms are big enough to support a large-scale

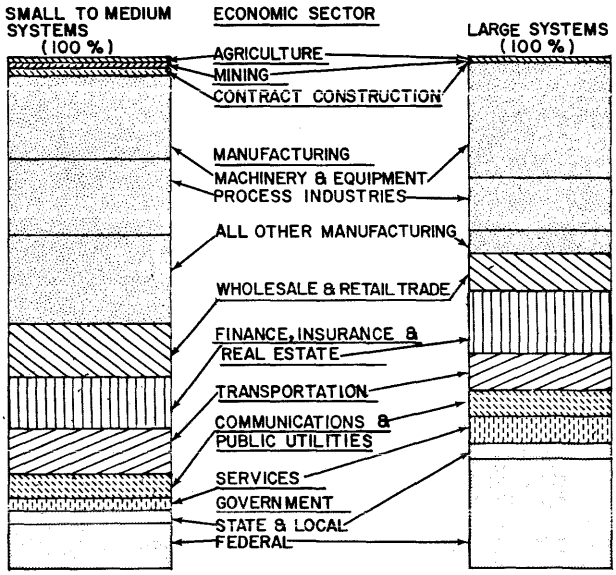


Exhibit 4. Use of Computer Systems in Ten Major Sectors of the U. S. Economy.

* The sectors considered include:

- Agriculture
- Mining
- Contract construction
- Manufacturing
- Transportation
- Communications and Utilities
- Wholesale and Retail Trade
- Finance, Insurance and Real Estate
- Services
- Government—state and local
- Government—federal

computing center. In certain sectors, the lack of computer applications concept, or suitable equipment, has prevented the development of applications. For example, the *Agriculture, Mining* and *Contract Construction* sectors of the economy make use of data processing systems to only a limited extent. In these industries, most of the work is done in the field, and the lack of availability of ruggedized, mobile computer systems which can operate under such field conditions have hindered the development of applications for these sectors. These industries also have a limited number of extremely large firms or organization units which can make effective use of large-scale processing systems. It is of interest, however, that the *Agriculture, Mining* and *Contract Construction* fields have all been marked by extensive use of materials handling automation.

The *Manufacturing* segment of the economy is the largest single industrial user of data-processing equipment. Computers are utilized in a wide range of standard functional areas such as production planning, financial and account processing, warehouse and inventory control, and support of research and development data analysis and calculations. Computers are also being applied to such advanced applications as real-time process control and numerical machine tool control.

Machinery and Equipment manufacturing forms a major sub-section of the manufacturing sector of the economy. This industry is composed of over 20,000 companies manufacturing a wide variety of mechanical and electrical systems, subsystems and components, ranging from carbon resistors to huge hydroelectric turbines and airplanes. Computer use in this sector is extremely heavy for scientific engineering calculations, as well as for the standard business functions. This industry is also beginning to apply numerical machine tool control systems. Over 800 such systems are now in use for virtually all types of machine tools.

There are two types of numerical control systems:

> *Positioning or point to point control,* in which the tool, such as a drill, can be moved in place, but cannot be angularly displaced. The path taken by the tool between successive points and the work piece is not a consideration.
>
> *Contour continuous path control,* in which the cutting axis of the tool itself can be changed, under the direction of the tool's controller. The path taken across the work piece is subject to continuous precise control.

Although point-to-point numerical control systems can be manually programmed, there have been a significant number of applications in which the programs are prepared by an off-line computer. Numerically controlled contouring, however, does require a digital computer to provide for the preparation and coding of machine tool commands. Program controlled, multi-operation machines may also require computers to co-ordinate their operations on a real-time basis. The application of numerical machine tool control has increased with the development of standard automatic programs, such as APT and Autoprompt.

Process manufacturing industries are also large users of data-processing equipment. For example, petroleum manufacturers were one of the first industrial groups to develop computer techniques for determining optimal blend ratios and to mechanize their billing and sales analysis operations. One of the most significant computer application areas in the process manufacturing industries at present is the development of computers for real-time process control. This application has been developing at a slower rate than was originally predicted, but the number of units installed is significant (Table 1).

TABLE 1.

Process Control Systems Installed in the United States 1962

Economic Sector	*Estimated Number of Units Installed*	*Estimated Value of Systems Installations*
Manufacturing	77	$8,736,000
Chemicals	22	$2,641,000
Petroleum	31	2,405,000
Iron & Steel	11	2,350,000
Other	13	1,340,000
Communications and Utilities	29	$5,421,000
TOTAL:	106	$14,157,000

Process control systems perform one or more of the following functions in the operation of a complete process:

- Data reduction
- Data logging
- Complete data control

If the system performs all three functions, it is termed a closed-loop operation. Several of the process control systems indicated in Table 1 perform only a data reduction or a data logging function. At least half of the units do involve a general purpose digital computer as part of the process control system.

The *Transportation industry* has generally not been a major use of data-processing equipment. Some of the larger railroads originally installed digital computers for accounting and financial operations, as well as for processing of freight traffic and rate calculations, in the late 1950's, but this application was never completely developed by the smaller railroad systems. More recently, railroads have been experimenting with digital data transmission operations. However, the present state of railroad finances has prevented any material development of this application. The largest segment of computer users within the transportation industry today are the airline companies. Almost every major airline firm is developing, or has already installed, a mechanized airline reservation system to maintain automatic count of seat inventories and availability. Some of these systems are also used for processing flight forecasts, weather information, manifest and passenger lists, schedules of operations, and management control reports. Approximately 11 major airline reservation systems will be installed by the end of 1963.

The Communications and Public utilities sector of the economy has also not made *significant* use of data-processing equipment. Most of the larger public power utilities have applied data-processing equipment for accounting and billing operations. Some of these organizations are now either installing, or considering the installation, of process control systems for real-time control of load assignment and dispatching. The major communication firm in this country, American Telephone & Telegraph, has always made extensive use of special-purpose data-processing systems for switching and network control. These applications have generally been specially engineered devices more properly identified as telephone switching and support gear than data-processing equipment, per se.

The *Wholesale and Retail Trade Sector* has been a fairly substantial, but not overly aggressive, user of small to medium class computer systems. Most of these units have been installed in the wholesale segment of the industry, for inventory control, and accounting operations. The automation of retail operations has been under consideration for several years, but most of these applications have been limited to the use of punched cards, punched tags and punched paper tape activated by cash registers as mechanized input/output devices. Larger department stores (such as Macy's and Gimbel's) as well as major mail order houses, are users of data-processing equipment for a wide variety of functions, including billing and accounting, integrated order processing and warehousing inventory control. However, the wide range of retail organizations in this country have not been affected by the application of data-processing equipment. Some consideration is now being given to the mechanization of the check-out operations in supermarkets with complete sales information prepared at the check-out station and transmitted directly to a control data-processing system. An important application to the retail industry is now under development. Credit Bureau operations, which are a basic element in retail store activities, have generally been performed manually, as a service in a regional area. The application of a digital computer system, to maintain credit references on all individuals in a regional area has been recently tested on the west coast.

The wholesale and retail industries have been slow to apply data-processing techniques primarily because of the unavailability of equipment geared specifically to their economic and operational requirements. Those organizations large enough to support tab equipment for their operations have recently turned toward the application of small class systems (such as the IBM1401 and the Remington Rand 1004), but the wholesale and retail field remain an area where the computer application development process has not moved at a rapid rate.

The *Finance, Insurance and Real Estate field* has been steadily applying the concepts of data-

processing technology to their operations. The *Insurance* field has been one of the most prolific users of data processing, with industry interest in computers, starting with the original UNIVAC development after World War II. The industry is a large user of medium and large tape-oriented computers. *Banks* have also been stepping up their application of computers, primarily for real-time systems in support of demand deposit operations. The applications of computers in the banks have been spurred by the development of Magnetic Ink Character Recognition (MICR) as a standard method of identifying and processing checks. Stock and commodity brokers have also been utilizing data-processing equipment. Recently the major stock markets have begun to install large-scale processing equipment to handle stock price changes and transactions.

The *Services industries* have also begun to increase their application of data-processing equipment in the last two to five years. An entirely new service of providing data-processing facilities on call has developed. Mechanized credit bureaus have already been mentioned as an application area which has just started to develop. Hospitals have also begun to turn to the use of small-scale computers for accounting and billing operations, as well as for mechanized information retrieval and processing of clinical information. Information service centers are also now appearing, based on the application of digital computers and large capacity memories to provide efficient processing and retrieval of information of a specialized nature for a fee.

The *Government* Sector has always been an extremely large user of data-processing equipment, although State and local Government utilization has been negligible. The Federal Government has applied computers to the widest possible range of functional areas, from the standard accounting activities to extremely complex systems for the processing of intelligence information (Exhibit 5).

The range of applications of computers and the extent of penetration in the U.S. economy is surprising. However, the number of segments of the economy in which applications are yet to be developed are even more intriguing. Any list of computer applications would be arbitrary, representing the bias of a particular

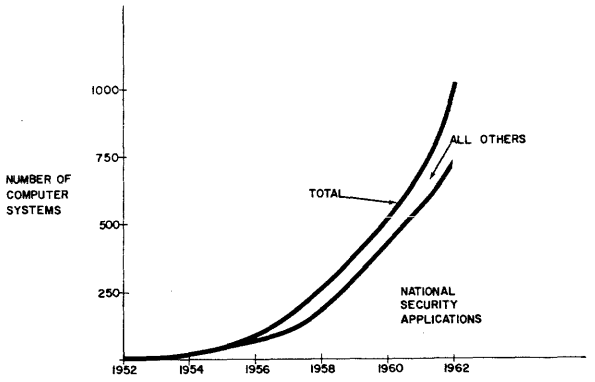


Exhibit 5. Number of Computer Systems Installed in the Federal Government.

orientation or discipline. However, a general list might include the following:

- Standard Financial and Accounting Operations
- Inventory Management Control and Analysis
- Processing of Scientific Engineering calculations
- Simulation and Gaming
- Process and Numerical Machine Tool Control
- Information Storage and Retrieval
- Management Planning and Strategic Control
- Intelligence Processing

Specific applications could also be defined, but they tend to be a subject of the broader application areas identified above. For example, PERT and critical path method applications, as well as the use of linear programming algorithms, may be classified under the general heading, "Management Planning and Strategic Control Operations." Utilizing these applications categories, we can arrive at some conclusions as to the extent to which various applications have been implemented in the 10 sectors of the United States economy previously described (Exhibit 6). Certain sectors of the economy, primarily the Federal Government and, to a lesser extent, the Manufacturing segment, have been developing a wide range of computer applications. Certain standard classes of applications, such as financial and account processing, have been

SECTOR		FINANCIAL AND ACCOUNTING	INVENTORY MANAGEMENT & ANALYSIS	PROCESSING OF R & D DATA	SIMULATION AND GAMING	PROCESS CONTROL	PRODUCTION CONTROL & PLANNING	INFORMATION STORAGE & RETRIEVAL	EXECUTIVE PLANNING & STRATEGIC CONTROL	INTELLIGENCE PROCESSING
AGRICULTURE MINING, AND CONTRACT CONSTRUCTION		A	A			A	A		A	
MANUFACTURING	MACHINERY AND EQUIPMENT							A	A	
	PROCESS CONTROL		A			A	A	A	A	
	OTHER							A	A	
WHOLESALE AND RETAIL TRADE		A	A					A	A	
FINANCE, INSURANCE, AND REAL ESTATE								A	A	
TRANSPORTATION			A		A	A		A	A	
COMMUNICATION AND PUBLIC UTILITIES			A			A			A	A
SERVICES		A		A	A			A	A	A
GOVERNMENT	STATE AND LOCAL							A	A	
	FEDERAL									

NOTATION:

- WIDE USE OF APPLICATION (MORE THAN 35% OF FIRMS)
- MEDIUM USE OF APPLICATION (10%-35% OF FIRMS)
- LITTLE USE OF APPLICATION (LESS THAN 10%) OF FIRMS
- A — APPLICATION AREAS WHERE GREATER USE OF COMPUTERS COULD BE MADE

Exhibit 6. Computer Applications in Various Sectors of the United States Economy.

utilized by organizations in every sector of the economy. However, other classes of applications, such as information storage and retrieval, and process control remain generally undeveloped.

We have surveyed the range of computer applications in the United States and given some semblance of structure to the application of these systems. We now turn to a more critical analysis of the applications development process, in order to assess the underlying factors in the success or failure of the development of new applications.

IV. A CRITICAL EXAMINATION OF THE APPLICATION DEVELOPMENT PROCESS

The previous analysis has demonstrated the wide range of computer applications, and the degree of use of these applications in various sectors of the United States. These applications have not all appeared at the same time, nor have they all been pursued with equal vigor. The evolution of computer applications during the last decade should be of deep interest to the industry. The dynamic forces which speeded up some applications and slowed others must be identified and understood in order to plan for the future.

Theoretically, the development of a computer application involves a fairly simple and straightforward process (Exhibit 7.1). A user develops an intial concept, and proceeds to select and plan for the installation of an appropriate hardware system and the necessary "software" and applications programs. In the event that either or both of these elements (hardware and programs) are not available, or are unsuited to the original concept, the concept is modified or discarded. If the required elements are available or can be obtained, the hardware is installed and programmed, resulting in (if all goes well), a successful application. It is fairly clear that certain critical stages exist in the applications development process, involving complex inter-relationships among:

- Concept development and planning
- Hardware selection and installation
- Program development and implementation

This applications development process is much less straightforward in real life. For a greater part of the last ten years, a majority of users were not technically capable of realistically developing new applications concepts.*

However, these concepts are an essential part of the creation of a market for computer systems. In order to generate orders for new installations, the computer manufacturers had to depend on its sales force. This sales role complicated and modified the Applications Development process (Exhibit 7.2). Thus the (computer manufacturers') sales force became engaged in presenting and generating new applications concepts as part of the successful procedure for selling the hardware system. Certain manufacturers, in recognizing this new role, began to systematically develop applications concepts to be used as part of the marketing package. The introduction and use of the *sales* force as a critical part of the applications development process was a very real, but often neglected, aspect of the computer industry in the last decade.

In order to appreciate the significance of the sales role, one must understand the basic marketing incentives involved. The salesman is re-

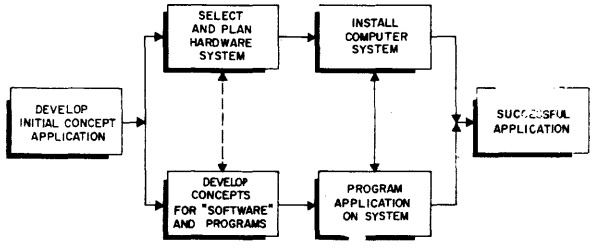


Exhibit 7.1. Theoretical Evolution of Application

* This situation is fortunately changing as a number of users have begun to develop their own technical staffs. However, the majority of the small to medium size users have not, and will probably never be able to support this type of technical skill.

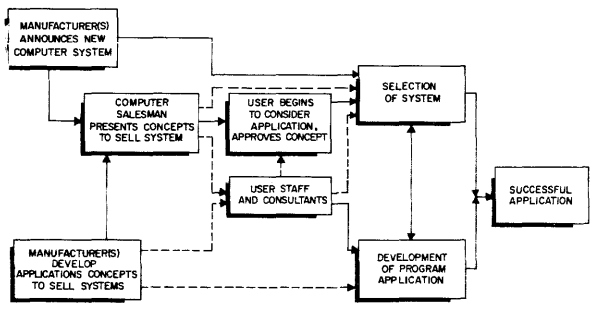


Exhibit 7.2. Evolution of Application Stressing Sales Role.

warded on the basis of the number and size of the installations of his *own* company's systems which his applications can generate. He is therefore forced to oversell those applications which are best suited to his own firm's commercially available product line, and undersell all other applications. In addition, he must keep a weather eye alert so that the application is best suited to his own firm's products as opposed to those of the competitors. He might even suggest and sell certain types of applications programs and software, especially geared to his own firm's equipment. This bias would not be terribly important if computer systems were designed completely on the basis of user needs. However, in the last decade, many computer systems were designed primarily by engineers intent on advancing their own view of the state of the art, with little or no awareness of the ultimate user application requirements. Thus, the applications development process was often not efficient, leading to bias in the development of applications, and a large number of failures.

These remarks did not apply to one particular class of user—the military departments of the Federal Government. The growing threat to national security, coupled with an increasing budget, made it possible for military agencies to introduce a stage in the applications development process which was generally not feasible for most industrial users (Exhibit 7.3). In essence, military users could, and often did, pay for the development of advanced computer systems and programs when none were commercially available. The military user was less

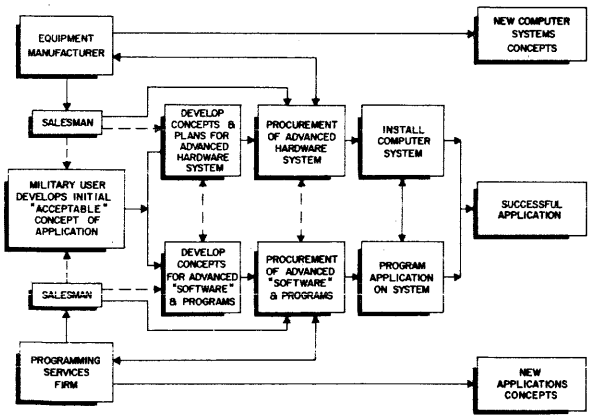


Exhibit 7.3. Evolution of Advanced Military Application Stressing Procurement Effects.

affected by the currently available product line and sales force than were industrial users. The array of military applications concepts were generally wider and more spohisticated, embracing applications for the over-all control of weapons, and forces, and for sophisticated processing of qualitative (intelligence) information.

However, systems manufacturers and programming service firms, in recognizing the growing special systems market, also introduced salesmen to exploit certain applications concepts involving skills and capabilities which they could supply.

This military systems application development process was made much more complex than in the typical industrial cycle because of the many different organizations and agencies involved. There were many more military system suppliers than commercial computer manufacturers. In fact, the computer manufacturer usually separated his military systems work from commercial computer development and sales efforts, resulting in even greater splits in emphasis. Programming services firms and a variety of "not-for-profit" systems analysis and development organizations also became involved in the applications development cycle.

The development of applications in military agencies of the Federal Government was also aided by a significant "bandwagon factor." New applications developments in one service often caused the separate generation of similar or slightly modified developments in other services.*

For example, command control applications were developed separately by the Air Force, Army, and Navy.

The military systems development process also produced an interesting side-effect. Entirely new computer systems and applications concepts were generated which ultimately found their way into the industrial applications development process.

Other factors affected the development of computer applications in the United States. One interesting example, from the standpoint

* This process was not necessarily inefficient. These parallel efforts often increased the rate of tchnological development by allowing more than one set of firms to generate new approaches to the same problem.

of future planning by manufacturers, is the "Landslide" effect which can occur in a specific industry or functional area (Exhibit 7.4).

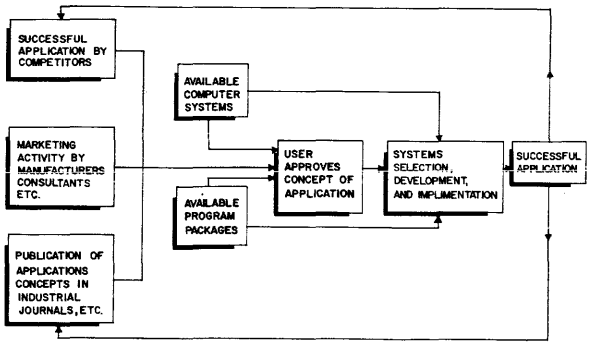


Exhibit 7.4. Expansion of Application in A Specific Industry or Functional Area, Illustrating "Landslide" Effect.

For example, an early area of computer applications was for Insurance firms which had a large degree of clerical operations easily convertible to machine functions. When the initial applications of these computers in major insurance companies demonstrated that EDP was both economical and extremely efficient compared to the old manual functions, many other insurance firms began to follow suit. This, in turn, caused other insurance firms who had been holding back, to begin to seriously consider the application of equipment simply because their competitors were doing so. In other words, successful application of particular functions in a competitor's operations has often resulted in a stampede by other firms in the same industry to develop and apply the same data-processing concepts in their own operations in order to continue to remain competitive. This cycle is strengthened by the effects of applications consultants and the appearance of articles in trade journals.

This competitive aspect can also affect the development of new applications. Once several competitors in the same industrial segment have machines, there is a tendency for each one to begin to *improve* his own system by broadening the number of applications and increasing the degree of mechanization in order to become more efficient than his competitors.

The competitive aspect in the development and implementation of applications of data-processing equipment is only one of the several factors involved in the growth of computer applications in the economy. Another factor stems from technological improvements which were made in computer equipment design starting in 1956. As new classes of equipment with particular capabilities became available, they opened up new areas of application which had not been previously considered, or were thought to be uneconomic. For example, the availability of the IBM 305 RAMAC equipment led many firms to begin to seriously consider warehousing and inventory applications based upon this medium-priced machine with extremely large memory, and efficient access time.

The applications development process in the last decade was the result of many different factors. However, two key elements

- The role of the marketing force in generating new applications concepts
- The role of the military in underwriting the development of hardware and computer programs

were, perhaps, the most significant.

IV. SUMMARY AND CONCLUSIONS

In assessing the application of data-processing equipment in the United States in the last few years, it is quite clear that computers have had a significant impact in all areas of the economy. However, an examination of the types of computer applications suggest that computers have been implemented for functional needs on only a piecemeal basis.

The application development process which led to the present situation was explored in some detail, pinpointing critical elements and factors in the cycle. The issue which should now be examined is the effectiveness of this past development process and changes which should and will come in the future.

Viewed from the objective of the national utilization of computers as an economic and technical resource, it is probably safe to conclude that the development process was carried out efficiently. Most of the applications development costs for the economy were paid for by either the computer manufacturers (who stood the most to gain), or by the military (which the total economy supports). Those few computer manufacturers who understood and supported the development process effectively, were

quite successful; although most paid a very high price for technological improvement without reaping the rewards. Fortunately, the military developments were more than enough to offset this economic and technical waste.

However, the situation is now changing. Military expenditures for computer systems developments are becoming more selective. In addition, the services and OSD are beginning to utilize commercial equipment to a greater extent, and reducing their non-essential systems development commitments. The "bandwagon" effect is slowly grinding to a halt.

In addition, certain manufacturers are beginning to *finally* recognize the need for intense product-market planning in order to develop user-oriented systems. Specialized industrial segments such as newspaper publishing, credit bureaus, legal practice, hospitals, and supermarket operations are being given more attention. More efficient and flexible computer modules are being developed in order to provide the sales force with the flexibility to generate usable concepts, rather than adhere to the parameters of technically advanced machines There is also an increasing awareness of the need to develop new applications concepts and efficient software packages, as a basic component of the successful marketing program. These tactics all stem from a recognition of the underlying components of the applications development process, and an increasing awareness of the need for detailed and extensive planning.

The emphasis of the last ten years was placed upon technology and the justification of the basic need for computers. If the computer industry is to continue to grow during the coming ten years, the emphasis must be placed upon the development and implementation of a broad range of sophisticated applications.

AUTOMATIC PARAMETER OPTIMIZATION AS APPLIED TO TRANSDUCER DESIGN

Max Howell
Senior Analog Applications Engineer
Martin Company
Orlando, Florida

PHYSICAL SYSTEM

It was desired to design a transducer array consisting of a circular piston source with three concentric rings in the same plane and having the same center. The rings were to be adjusted in diameter and source strength to increase the directivity of the piston by narrowing the main beam and reducing the level of the minor lobes. The directivity of a piston (D_p) is given by

$$D_p = \frac{2J_1(u)}{u} \tag{1}$$

where

$$u = \frac{\pi d}{\lambda} \sin \theta$$

$J_1(u)$ = Bessel's function[1] of the first kind of order one,
d = diameter of the piston,
λ = the wavelength of the sound, and
θ = the angular direction with respect to the main axis.

The directivity of a ring is given by Bessel's function of the first kind of order zero, $J_0(u)$. If $y_1 (< 1)$ is the fraction of the piston source strength used for ring source strength and x_1 (> 1) is the ratio of the ring diameter to piston diameter, the resultant directivity of the piston plus the rings can be expressed by

$$D(u) = \frac{2J_1(u)}{u} + y_1 J_0(x_1 u) + y_2 J_0(x_2 u) + y_3 J_0(x_3 u) \tag{2}$$

For convenience we normalize Equation (2), by dividing by $D(0)$, so that we have the new function $N(u)$ where $N(0) = 1$. For the normalized form of the directivity we have

$$N(u) = \frac{1}{1 + y_1 + y_2 + y_3} D(u) \tag{3}$$

The range of values of u to be considered are from 0 to 16. The driving source to the piston and rings was considered to have the same phasing for simplicity. A consideration of phasing would have required the optimization of three additional parameters.

OPTIMIZATION PROCEDURE

In order to get the desired directivity, it was decided to construct a model having a fast crossing of the u (independent variable) axis and little overshoot. Since a highly damped second order system with the same initial condition is of this nature, it was used as a model. Now by solving for the normalized directivity $N(u)$ as a function of the u and comparing with the second order model $M(u)$, we have a u history of the error $\epsilon(u)$. If we integrate the absolute value of the function $\epsilon(u)$, we have a criterion to compare the desirability of solutions $N(u)$ for any given values of the parameters $(x_1, x_2, x_3, y_1, y_2, y_3)$. If we start with a fixed set of the values $(x_1, x_2, x_3, y_1, y_2, y_3)$ and adjust one of the parameters, say x_1 by a fixed amount δ, we have the new set of parameters $(x_1 + \delta, x_2, x_3, y_1, y_2, y_3)$. Now if we compare

the functions $\int |\epsilon(x_1 + \delta, x_2, x_3, y_1, y_2, y_3)| \, du$ and $\int |\epsilon(x_1, x_2, x_3, y_1, y_2, y_3)| du$, we can determine if the adjustment δ has improved $N(u)$. If $N(u)$ has been improved, we again make the adjustment δ and continue to do so until $N(u)$ is not improved. When $N(u)$ is not improved, we make the adjustment $-\delta$, and continue to do so until $N(u)$ is not improved. Then we again make the adjustment δ, and switch to adjustments in another parameter, say y_1. We go through the same process and when finished, we adjust another parameter. After all parameters have been adjusted, we repeat the cycle starting again with x_1. This is continued until we no longer have any improvements in $\epsilon(u)$. The resultant values of $(x_1, x_2, x_3, y_1, y_2, y_3)$ are now recorded and they establish a minimum of the function $\epsilon(u)$. Other minima of $\epsilon(u)$ can be found by starting with different initial conditions of the six parameters.

By implementing this procedure at 1 unit of u equals 5 milliseconds and a repetition rate of one solution per 80 milliseconds, time for convergence was in the order of one minute.

EQUATIONS AND MECHANIZATION OF SYSTEM

The equations and mechanization (Figure 1) for the ring contribution are as follows:

$$\begin{aligned} & x_i u \frac{d^2 J_0(x_i u)}{d(x_i u)^2} + \frac{dJ_0(x_i u)}{d(x_i u)} + x_i u J_0(x_i u) \\ & = 0, \; J_0(0) = 1 \\ & \qquad J_0'(0) = 0 \\ & \qquad i = 1, 2, \text{ or } 3 \end{aligned} \tag{4}$$

For the piston we have:

$$\begin{aligned} & u^2 \frac{d^2 J_1(u)}{du^2} + u \frac{dJ_1(u)}{du} + (u^2 - 1) J_1(u) \\ & = 0, \; J_1(0) = 0 \\ & \qquad \frac{dJ_1(0)}{du} = \frac{1}{2} \end{aligned} \tag{5}$$

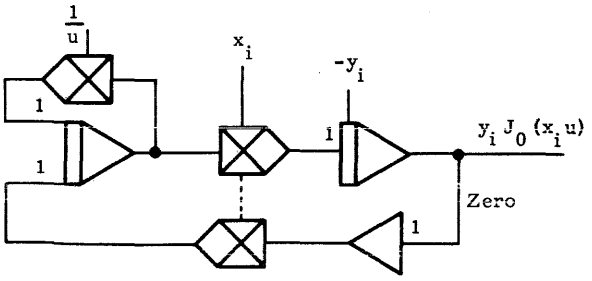


Figure 1. Ring Contribution.

However since we are interested in the function $\frac{2J_1(u)}{u}$, (Figure 2) we make the substitution $J_1(u) = uy_1(u)$.

$$\begin{aligned} & u \frac{d^2 y_1(u)}{du^2} + 3 \frac{dy_1(u)}{du} + u y_1(u) \\ & = 0, \lim_{u \to 0} \frac{dy_1(u)}{du} = 0 \\ & \qquad \lim_{u \to 0} y_1(u) = \frac{1}{2} \end{aligned} \tag{6}$$

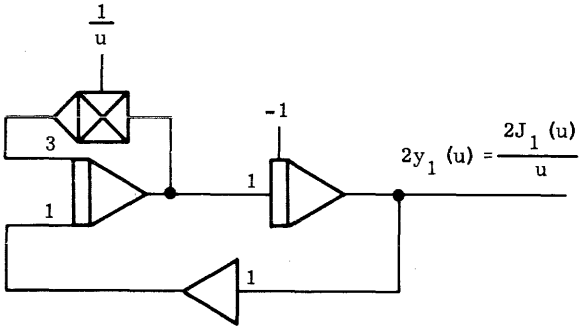


Figure 2. Piston Contribution.

The approximation $1/u = 10$, $(0 \leq u < 0.1)$ was used to achieve high resolution of the variable $1/u$, $(0.1 \leq u \leq 16)$. This improves the accuracy of the multiplications by $1/u$ at the expense of a slight error in $1/u$.

The mechanization of the normalized directivity with scale factors, etc., is shown in Figure 3.

LOGIC AND CONTROL CIRCUITS

Having now established the mechanization of the directivity as a function of the six parameters, we must determine a means to adjust the parameters so that directivity $N(u)$ will converge on the model $M(u)$. First we will establish the error criterion $\int |\epsilon| \, du$, which for a given run (N) will be called E_N (Figure 4).

The next step is to have a device to remember the previous error, E_{N-1}, so it can be compared with E_N to tell whether or not we are converging (Figure 5). In this case, a "track and hold" integrator[2] followed by a "hold and track" integrator will suffice. This is followed with a comparator so that small differences can be measured and also to give a digital type intelligence.

The third step is a circuit to tell the parameter, if it is moving in the wrong direction, to turn around and go the other way. It requires a remembrance of the preceding direction and a means of reversing this whenever E_{N-1}—E_N goes negative. This was mechanized as indicated in Figure 6. In the circuit of Figure 6, we compute the changes in the parameters.

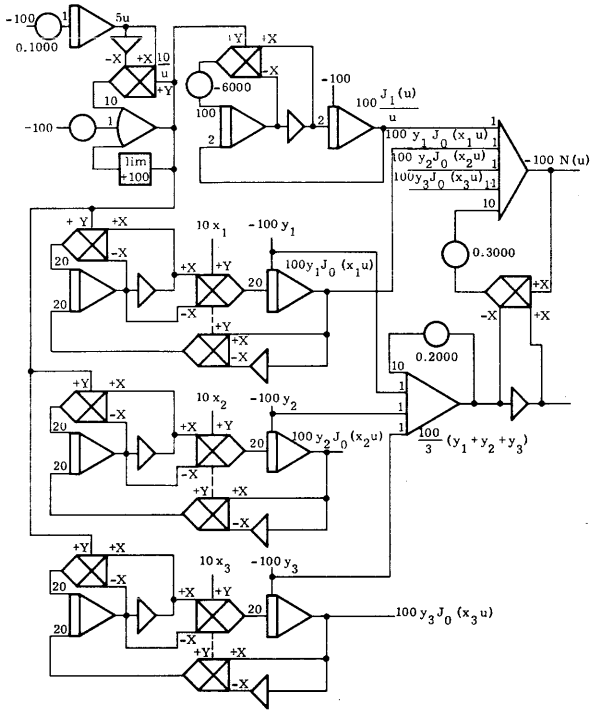


Figure 3. Normalized Directivity Simulation.

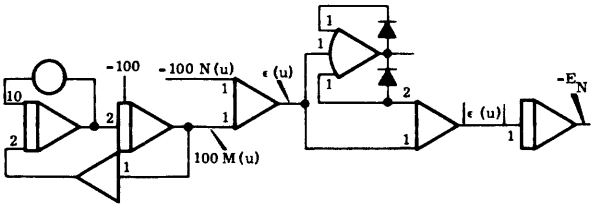


Figure 4. Error Criterion Generation.

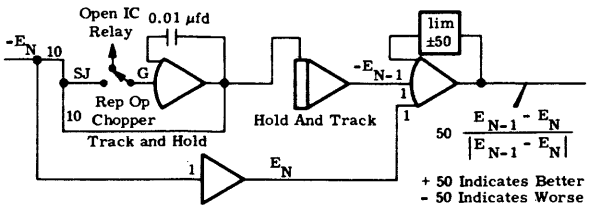


Figure 5. Error Memory and Comparator.

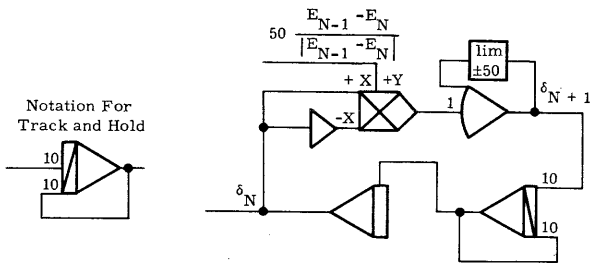


Figure 6. Parameter Increment Logic.

Next we need a circuit to switch from one parameter's optimization to another. It was decided the best time to do this is when a better result is computed after two worse results. The reason for this criterion is that if we initially started in the wrong direction the computer would be told to turn around and proceed until the next worse result is obtained; then, take a step back and change parameters. Before this could be done the value of $50\,\frac{E_{N-1}-E_N}{|E_{N-1}-E_N|}$ had to be remembered through the next hold cycle. Incidentally, all of the adjustments in the parameters were done during the hold (reset) portion and the error was computed in the track (operate) portion. This was mechanized as shown in Figure 7.

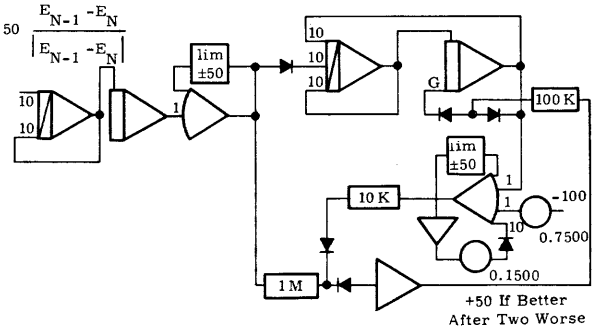


Figure 7. Parameter Sequence Logic.

An alternate method was to use digital logic for this application. This was less complex, but for this purpose was more susceptible to extraneous disturbances and therefore gave incorrect signals to the parameter changers. This circuit is indicated in Figure 8. This method would have been more desirable had the logic components been synchronous (clock driven). In this case there would be less chance for external interruption.

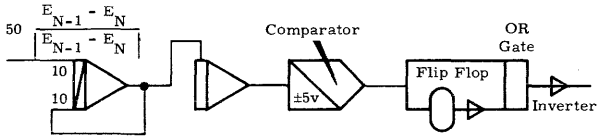


Figure 8. Parameter Sequence Logic (Digital).

Now it is necessary to count these "better after two worse" signals from one to six, and then reset the counter to one and start over. Each state of the counter will determine which parameter is receiving plus or minus voltages from the "change directions if worse" circuit. The parameter that is receiving these voltages will accumulate them; otherwise, it will hold its present value. This was accomplished by means of a six position stepping relay. The advantage of the relay is that when a parameter was not being optimized there was absolutely no bias feeding into its accumulator. A ring counter in conjunction with six diode gates was also tried for this purpose, but proved undesirable because of the gate offsets feeding into the accumulators. The stepping relay was acceptable for 80 millisecond track and 40 millisecond hold periods. However, if a shorter rep-op cycle had been used, high speed switching would have been necessary and parameter drift may not have been such a problem. The stepping relay mechanization is shown in Figure 9.

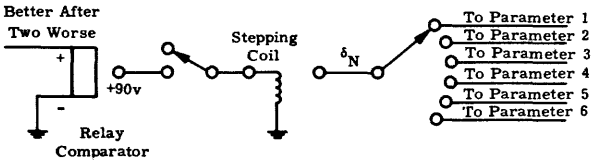


Figure 9. Parameter Sequencer.

The final link in the optimization is the accumulator (Figure 10). Two methods were tried in this case, the accounting circuit (A) and a real time integrator (B). It was thought that the accounting circuit would be desirable because it would store the changes in the parameters discretely. However, the accounting circuit will tend to converge or diverge by itself without any inputs. This is because if the loop gain is greater than unity, each time the information is recycled it will be slightly increased. Furthermore, if the loop gain is less than unity when the information is recycled, it will be slightly decreased. Therefore, the real time

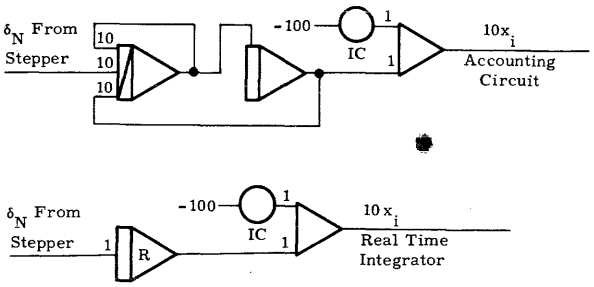


Figure 10. Parameter Accumulators.

integrator approach was tried, and proved satisfactory. The circuits for these methods are shown below.

Now by putting all of these components together, we are able to optimize the given system. The complete logic and control diagram is provided in Figure 11.

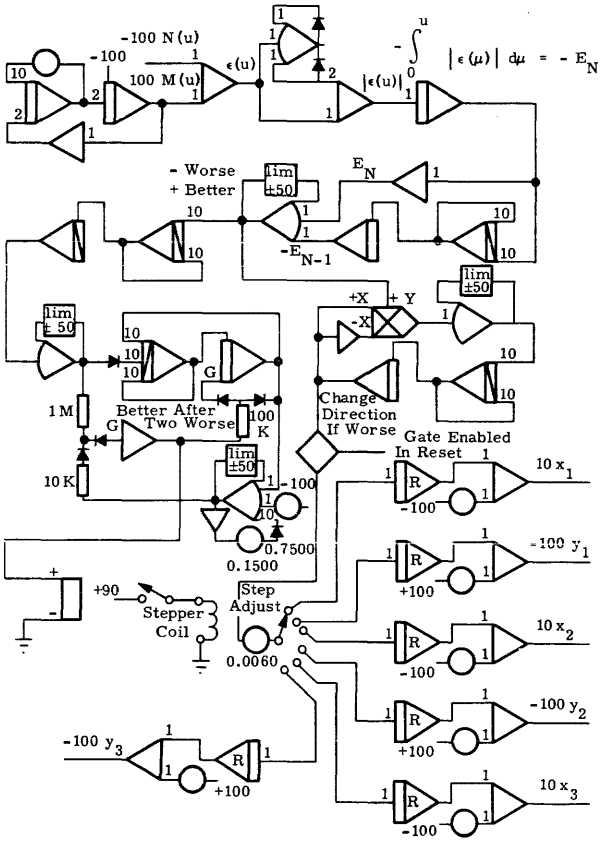


Figure 11. Combined Logic Diagram.

RESULTS

Figures 12, 13 and 14 illustrate the results obtained from the described method. Three cases are shown, each of which represents a

given set of initial conditions of the six parameters. There are two graphs for each case. The first shows the directivity, error criterion, and model before optimization (Figures 12A, 13A, and 14A). The second shows these same functions after optimization (Figures 12B, 13B, and 14B). The graphs were plotted in the normal computing mode using the values of the parameters obtained in the repetitive-operation mode. Although the x's are initially equal, which is physically impossible, the equations are still good and the final configurations are realizable.

REFERENCES

1. KENNETH S. MILLER, "Engineering Mathematics," Rinehart & Company, Inc., New York (1956) : 116-123.
2. JACK M. ANDREWS, "Mathematical Applications of the Dynamic Storage Analog Computer," Proceedings of the Western Joint Computer Conference (1960) : 119-131.

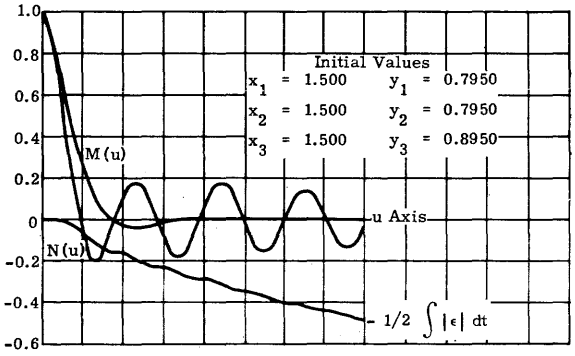


Figure 12A. Case I, Before Optimization.

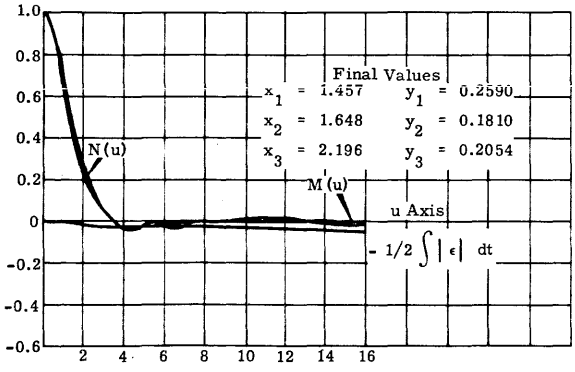


Figure 12B. Case I, After Optimization.

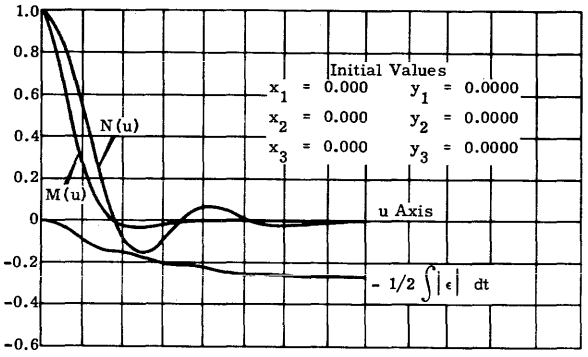


Figure 13A. Case II, Before Optimization.

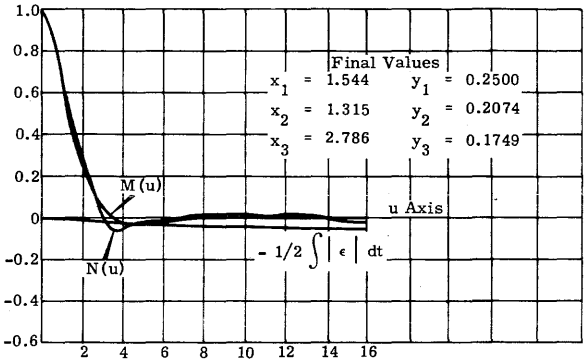


Figure 13B. Case II, After Optimization.

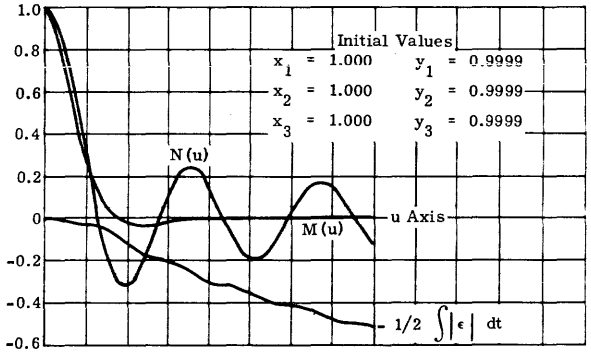


Figure 14A. Case III, Before Optimization.

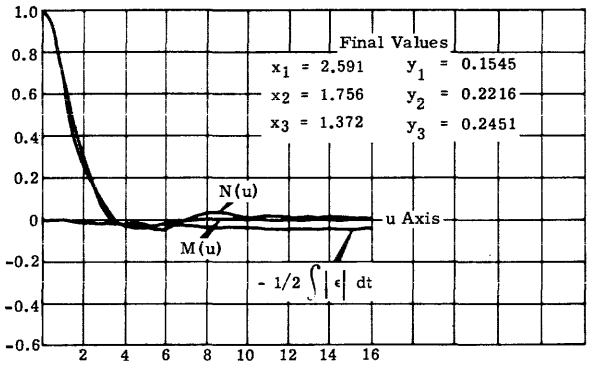


Figure 14B. Case III, After Optimization.

HYBRID COMPUTER SOLUTION OF TIME-OPTIMAL CONTROL PROBLEMS

Elmer G. Gilbert
Instrumentation Engineering Program
University of Michigan
Ann Arbor, Michigan

INTRODUCTION

Optimal control systems are of considerable practical and theoretical interest. Although solutions of certain optimal control problems have been known for many years, it is only recently that fairly general, rigorous solution techniques have been developed. Unfortunately, the computational aspects of these solution techniques still present formidable problems. The time-optimal problem which is treated in what follows has a very well developed theory. Our purpose here is to show the utility of hybrid computer techniques. Some of the programming procedures described may also be useful in the solution of other problems.

THE TIME-OPTIMAL CONTROL PROBLEM

It is assumed that the physical system to be controlled (hereafter called the plant) satisfies the following system of first order linear differential equations.

$$\dot{x}_i = \sum_{j=1}^{n} a_{ij}(t)x_j(t) + b_i(t)u(t), \qquad (1)$$

$$x_i(0) = x_i^0 \qquad i = 1, 2, \ldots n$$

To simplify notation vector-matrix notation is employed.

$$\dot{x} = A(t)x + b(t)u, \; x(0) = x^0 \qquad (2)$$

Thus x and $b(t)$ are n vectors and $A(t)$ is an uxn matrix. For mathematical convenience it is assumed that $A(t)$ and $b(t)$ are continuous. Given an initial state $x(0) = x^o$, the scalar control signal $u(t)$ must be selected to achieve a specified motion $x(t)$ for $t > 0$. To make the problem of practical interest it is required that $u(t)$ be piecewise continuous and

$$|u(t)| \leq 1. \qquad (3)$$

A control which satisfies these two conditions is said to be admissible.

The time-optimal control problem is stated as follows: Given the plant (2) with initial state $x(0) = x^o$ and a desired terminal state x^d, find an admissible control which makes $x(t) = x^d$ for the smallest possible t. If such an optimal control exists it will be called $u^*(t)$. The minimum time and optimal motion associated with $u^*(t)$ will be denoted by t^* and $x^*(t)$, i.e., $x^*(t^*) = x^d$. Our concern is the computation of $u^*(t)$.

The theory of the problem has been treated by many authors (see, for example, the article by LaSalle[1] or the book by Pontryagin et al[2]). The basic result of the theory is the following. If a $u^*(t)$ exists it is given by

$$u(t) = \text{sgn } v(t) \qquad \dagger \qquad (4)$$

where

$$v(t) = -(b, \xi) \qquad \dagger \qquad (5)$$

and $\xi(t)$ satisfies the adjoint differential equation

$$\dot{\xi} = -A'(t)\xi, \; \xi(0) = \eta \quad \dagger \qquad (6)$$

† The notation is conventional. sgn $v = 1$, $v > 0$; sgn $v = -1$, $v < 0$; sgn v undefined, $v = 0$. $(b, \xi) = \sum_{i=1}^{n} b_i(t)\xi_i(t)$. $A'(t)$ is the transpose of $A(t)$.

for some initial condition vector $\eta = \eta^*$. Equation (4) says that $u^*(t) = \pm 1$, i.e., it is a "bang-bang" control.

Since sgn v is not defined for $v = 0$, $u^*(t)$ is undefined unless $v(t) = 0$ only at isolated instants of time (the switching times). If for all $\eta \neq 0$, $v(t) = 0$ only at isolated instants of time, the system (2) is called "normal" or equivalently, "completely controllable." LaSalle considers the normality condition at length. If A and b are constant he shows that (2) is normal if the vectors $A^k b$, $k = 0, 1, \ldots, n\text{-}1$, are linearly independent. In what follows it is assumed that (2) is normal.

The fact that η^* is not explicitly defined is a disadvantage in practical computation of $u^*(t)$. Each value of η produces a control $u = \text{sgn } v$ which, when applied to the plant equations (2), produces a time-optimal motion. In fact, when the set of all values of η is used, the set of all time-optimal motions is generated. The computational problem is to select the particular optimal motion $x^*(t)$ which passes through the desired terminal state x^d. This can be done by searching through all value of η until an optimal motion through x^d is obtained. Since for each η, (2) and (6) must be solved together, it is impossible to consider all η. Instead, a finite set of η values must be used with the hope that one of the optimal solutions will pass close to x^d. A better approach is to formulate an algorithm which generates a sequence of η's $\{\eta^0, \eta^1 \ldots .\}$ which converges to η^*. In later sections both this approach and the search approach are programmed on a hybrid computer.

Before considering the hybrid computer a few additional facts should be noted. First, the magnitude of the η vector is immaterial. This is so because v is proportional to the magnitude of η but $u^*(t)$ depends only on the sign of $v(t)$. Thus the magnitude scaling of η (and ξ) is immaterial. Finally, normality implies that if there is an optimal motion through x^d it is unique and so is the associated control. Thus even though there are many η^* which produce the same optimal control, there can be only one optimal control.

THE HYBRID COMPUTER

The hybrid computer employs both analog and digital computing devices, which may conveniently be organized in three categories: analog, analog-digital, and digital, according to the kind of signals with which they deal. For the class of problems considered here it is most natural to let the analog devices integrate the differential equations and to reserve the digital devices for problem control. Because of the rather modest requirements on the digital portion of the system it can consist of rather simple logic elements.

Figure 1 gives symbols for the basic computer elements. Conventional analog elements are not shown. To make signal designations clear, light lines and lower-case letters are used for analog signals and heavy lines and upper-case letters are used for logic signals. A denotes the complement of A. Elements (a) - (f) are analog-digital and elements (g)-(k) are digital.

Integrator (a) and (b). Each integrator is separately controlled through its operate (O) and hold (H) inputs.

$A = 0$: $e_0 = -e$, initial condition mode, B has no effect

$A = 1, B = 0$: $e_o = \alpha \int_o^t e_1 d\sigma - e$, operate mode

$A = 1, B = 1$: $e_0 =$ constant, hold mode

Switch (c) single throw, (d) double throw.

$A = 0$: $i = 10e_2$, $e_0 = -e_2$ (for (c), $e_2 = 0$)

$A = 1$: $i = 10e_1$, $e_0 = -e_1$

Comparitor (e). The comparitor has hysteresis with dead zone $\pm \epsilon$. $B = 1$ locks the comparitor output A at the level present when $B = 0 \rightarrow 1$.*

$e_1 + e_2 > \epsilon$: $A = 1$

$e_1 + e_2 < -\epsilon$: $A = 0$

$|e_1 + e_2| < \epsilon$: $A = 0$ or 1, depending on past history of $e_1 + e_2$

Analog Memory or Track-Transfer (f)

$B = 0$: $e_0 = e$, initial condition mode, A has no effect

$B = 1$, $A = 0 \rightarrow 1$: take $e_1 + e_2$ and transfer to e_0, hold e_0 constant until next $A = 0 \rightarrow 1$.

AND Gate (g)

$D = A \cdot B \cdot C$, $D = 1$ only if $A = B = C = 1$

OR Gate (h)

$D = A + B + C$, $D = 0$ only if $A = B = C = 0$

Flip-flop (i). The inputs A and C cannot simultaneously equal 1

$A = 1, C = 0$: $D = 1$, set flip-flop

$A = 0, C = 1$: $D = 0$, clear flip-flop

$A = C = 0$: $D = 1, 0$, store 1 or 0

$A = C = 0, B = 0 \rightarrow 1$: reverse D

* The notation $B = 0 \rightarrow 1$ is used to indicate the transition of B from logic 0 to logic 1.

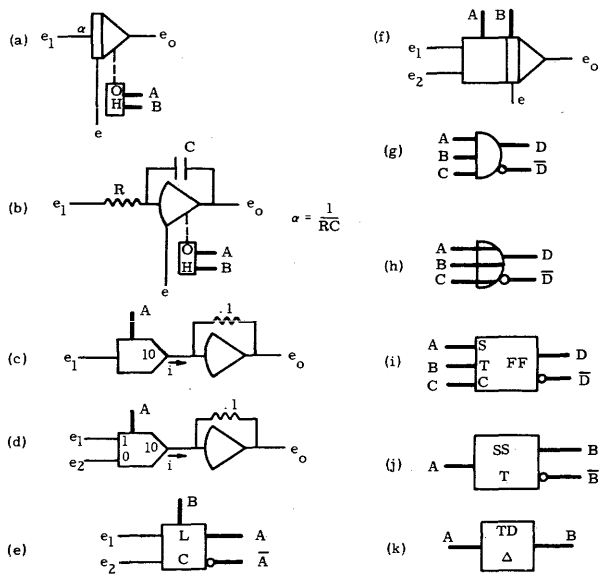


Figure 1. Computer Symbols.

Single-shot (j)

$A = 0 \rightarrow 1$: $D = 1$ for T seconds, otherwise $D = 0$

Time-delay (k)

$B = A$ delayed by $\triangle$ seconds

Some additional comment on the general function of these elements should be made. The comparitor dead zone $\pm\epsilon$ is small but necessary since it is the only way to assure uniformly fast and positive transition of the comparitor output. The track-transfer memory element combines in a single unit the function of two track-hold memory units connected in cascade, along with initial condition provisions. For ease of programming, each element with a logic output also has the complemented output. Finally, it should be noted that the elements are not controlled by a master system clock. This simplifies programming and prevents possible errors due to the quantizing of time.

Additional computer equipment would include input control buttons, display lights, shift registers, clocks, counters, problem check provisions, etc. However, it is not necessary to go into these details here.

THE COMPUTATION OF OPTIMAL SOLUTIONS AND THE REACHABLE SET

Figure 2 shows the computer programming for generation of time-optimal solutions. For brevity the details of programming in blocks I and III are not shown. The integrators must all be started together; thus the operate (O) inputs on blocks I and III are joined. Each time $A = 0 \rightarrow 1$ an optimal solution $x(t)$ is generated. By driving A from an oscillator and running the integrators on a fast time scale, conventional repetitive operation is obtained. By systematically changing η a search for η^*, which produces the desired optimal control u^* and motion x^*, can be made.

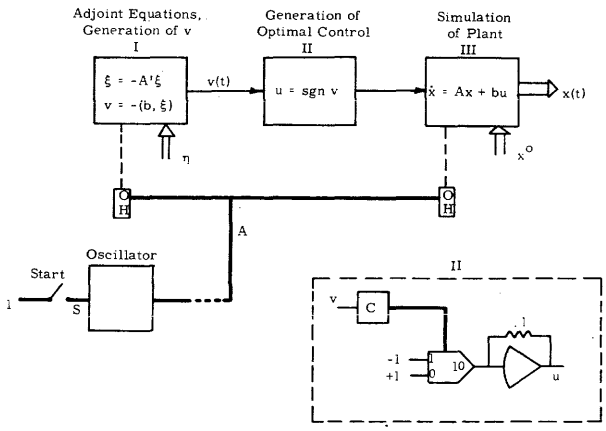


Figure 2. Computer Program for Generation of Optimal Solutions.

Actually, if all η values are to be scanned, much more general information concerning the control of (2) may be obtained. In particular, the reachable set of (2) may be determined. The reachable set $R(t, x^o)$ is the set of all states which may be obtained at time t (starting from x^o at time $t = 0$) by means of admissible controls. Thus $R(t, x^o)$ determines the extent of possible motion with admissible controls. It can be shown that the set of all time optimal solutions delineates the boundary of $R(t, x^o)$.[3] Thus to obtain a boundary point of $R\ (T, x^o)$ (T is a particular value of t) an η is chosen and a solution from Figure 2 generated. At $t = T$, $x(t)$ is the boundary point. By taking a suitable set of η's enough boundary points are obtained to define $R\ (T, x^o)$. Let us determine a computer program for the second order case ($n = 2$).

First, a procedure for scanning through a set of values for η is required. Since the magnitude of η is immaterial, only its direction is essential. Thus take $\eta_1 = \sin\theta$ and $\eta_2 = \cos\theta$. By scanning θ in the interval $(0, 2\pi)$ a suitable set of time optimal solutions is generated.

Figure 3 shows one procedure for generating *sin* θ and *cos* θ. The equation

$$\ddot{z} + z = 0, \quad z(0) = 1, \quad \dot{z}(0) = 0 \qquad (7)$$

is solved, producing the functions $z = \cos \tau$ and $-z = \sin \tau$ which, when held at $\tau = \theta$, produce the desired outputs. Operation of the various computer elements should be clear from the timing diagram (a heavy line in the diagram indicates the presence of logic 1). The outputs cos θ and sin θ are available $\tau = \theta$ seconds after the input command B is initiated and must be read out before B returns to logic zero.

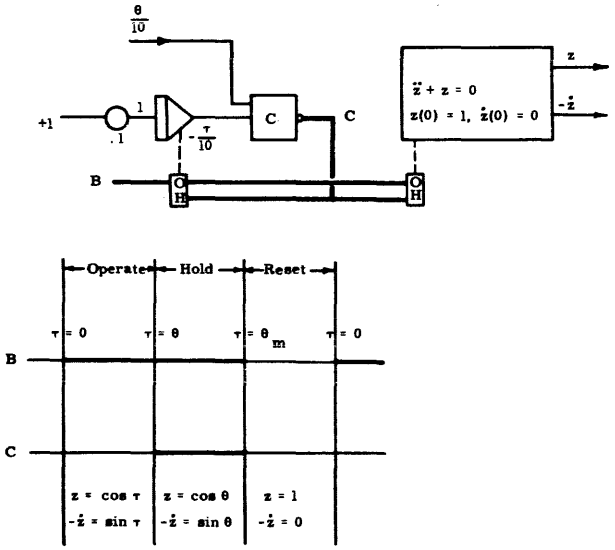


Figure 3. Computer Program and Timing Diagram for Function Generation.

Figure 4 shows the program for generating the boundary points of the reachable set $R(T, x^o)$. Before the start of the computation ($S = 0$) the oscillator is turned off ($A = 0$) and the integrators in Figure 2 are in the reset mode. Since the single-shot is in its rest state, $B = 0$ and the function generator is in its reset state. Thus $\eta_1 = 0$ and $\eta_2 = 1$. ($\theta = 0$). At $S = 0 \rightarrow 1$ the first run begins since then $A = 1$. At $t = T$ the comparitor output $D = 0 \rightarrow 1$, triggering the memory units and transferring the first boundary point to their outputs. Since the outputs are held until $t = T$ in the next run, there is ample time for read out (an x, y recorder or oscilloscope depending on the time scale). In the interval (0, 1) the single-shot

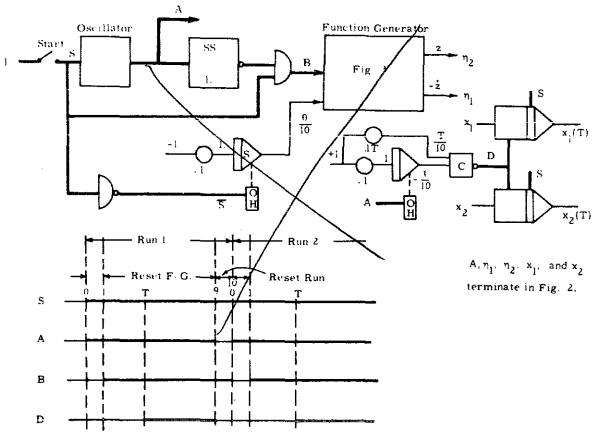


Figure 4. Program and Timing Diagram for Generating Boundary Points of $R(T, x^o)$.

causes $B = 0$ and the function generator is reset. At time $1 + \theta$ the θ for the next run is set in. By generating θ in a slow time integrator which is controlled by S, θ is allowed to increase slightly from run to run. When $\theta = 2\pi$ the computation is complete. In the interval (9, 10) of the first run the integrators are reset for the next run, the new values of η_1 and η_2 being used. In the second and subsequent runs the above pattern is the same.

Extension of the above program to higher order systems is fairly straightforward. The main complication is the scanning of the higher dimensional η vectors. Also the number of runs must be increased greatly because of the higher dimensionality of the $R(T, x^o)$ boundary. In fact, finding a suitable way of storing the boundary points may be more of a problem than their computation.

THE REGULATOR PROBLEM, ITS ITERATIVE SOLUTION

In this section an iterative procedure is described for solving the time-optimal regulator problem where the terminal state $x^d = 0$. A sequence of vectors $\{\eta^o, \eta^1, \ldots .\}$ is produced which converges to η^*. Additional detail and proofs are given in the paper by Neustadt.[3]

First a condition which η^* must satisfy is given. In what follows η is always taken so that

$$(\eta, x^0) > 0 \qquad (8)$$

For each η there is always a corresponding $v(t)$ defined by (5) and (6). Thus the function

$$f(t, \eta) = -\int_0^t |v(\sigma)|\, d\sigma + (\eta, x^0) \qquad (9)$$

is always defined. Because of (8), $f(0, \eta) > 0$. Also $f(t, \eta)$ is clearly nonincreasing. In fact, if there is a time-optimal solution $f(t, \eta)$ eventually becomes zero. Let $t_s(\eta)$ be the time when $f(t, \eta) = 0$. Neustadt shows that the following facts are true: $t_s(\eta) \leq t^*$, $t_s(\eta) = t^*$ only if $\eta = \eta^*$, and grad $t_s(\eta) = 0$ only if $\eta = \eta^*$. Thus by choosing η to maximize $t_s(\eta)$, $t_s = t^*$ and $\eta = \eta^*$. Since grad $t_s(\eta) = 0$ only if $\eta = \eta^*$ the method of steepest ascent is appropriate to maximizing $t_s(\eta)$. Thus

$$\eta^{j+1} = \eta^j + k \text{ grad } t_s(\eta^j) \qquad (10)$$

For $k > 0$ sufficiently small and $(\eta^o, x^o) > 0$ the sequence of vectors $\{\eta^o, \eta^1, \ldots\}$ converges to η^*.

Neustadt also gives a formula for determining grad $t_s(\eta)$. It can be shown that evaluating the formula is equivalent to the following[4]: take $u = \text{sgn } v(t)$ and solve

$$\dot{w} = A(t)w + b(t)u, \quad w(0) = x^0 \qquad (11)$$

from $t = 0$ to $t = t_s$, at $t = t_s$ run time backwards and solve (11) with $u = 0$, when $t = 0$ determine w and call it $\gamma(\eta)$, then

$$\text{grad } t_s = K(\eta)\, \gamma(\eta), \quad K(\eta) > 0 \qquad (12)$$

Since $K(\eta)$ is a positive quantity it is not necessary to determine its value in order to employ the method of steepest ascent. Instead of (10)

$$\eta^{j+1} = \eta^j + \rho\gamma(\eta^j) \qquad (13)$$

is used $\left(k = \frac{\rho}{K}\right)$.

To summarize, the iterative procedure consists of the following steps:

(a) Choose an initial $\eta^j j = 0$ so that $(\eta^j, x^0) > 0$
(b) Use (5), (6) and (9) to define $f(t, \eta^j)$
(c) Determine $t_s(\eta^j)$ from $f(t, \eta^j) = 0$
(d) Let $u = \text{sgn } v(t)$ and solve (11) from $t = 0$ to $t = t_s(\eta^j)$
(e) At $t = t_s(\eta^j)$ set $u = 0$, reverse time and solve (11) from $t = t_s(\eta^j)$ to $t = 0$
(f) At $t = 0$ set $w = \gamma(\eta^j)$
(g) Determine the next value of η^j from (13)

THE PROGRAMMING OF THE ITERATIVE SOLUTION

The steps in the iterative procedure are sufficiently involved that it is wise to break the programming down into a series of subproblems: the solution of (11), the control of (11) to produce $\gamma(\eta)$, the generation of t_s, and the implementation of (13).

Figure 5 shows the programming of (11). Block I, as before, generates $v(t)$ for a given η. The operate (O) input $F = 0 \rightarrow 1$ at $t = 0$ and $F = 1$ until $t = t_s(\eta)$. During the reverse time integration of (11) $v(t)$ is not required so during this time $F = 0$. Block II generates $u = \text{sgn } v$ for $t = 0$ to $t = t_s$ and $u = 0$ as t goes backward from $t = t_s$ to $t = 0$. The logic signal E, which is 1 for forward time and 0 for reverse time, exercises the desired control. When $E = 0$ both AND gates have 0 output causing both switches in II to be open. On the other hand, when $E = 1$, either one switch or the other is closed depending on the sign of $v(t)$. Block III solves (11), a typical integrator being shown in Figure 5. When $G = 0 \rightarrow 1$ the integrators begin to operate. From $t = 0$ to $t = t_s$, $E = 1$ producing forward time. At $t = t_s$, $E = 1 \rightarrow 0$ and the integrators begin to move backwards. When $t = 0$ again, the hold signal $H = 0 \rightarrow 1$. Thus $\gamma(\eta)$ is held at the integrator outputs as long as $G = H = 1$.

The control program for generating F, E, G, and H is shown in Figure 6. From what has been said it is clear that $E = F$. The desired sequencing of E, G, and H is shown in the timing diagram. The signals E and I (never 1 at the same time) are the inputs to the control program. When $I = 1$ the integrators in Figure 5 are reset in preparation for a run. The 0 stored in the flip-flop maintains this condition even after $I = 1 \rightarrow 0$. To produce a run, $E = 0 \rightarrow 1$ and remains at 1 until $t = t_s(\eta)$, when $E = 1 \rightarrow 0$ and stays at 0 until the next run. When $E = 0 \rightarrow 1$ the flip-flop is set and $G = 1$ until the flip-flop is cleared by I for the next run. The integrator, comparitor, and AND gate produce H. The integrator output is positive during forward time (t) and reverse time (t') until $t' = 0$ is reached, when the comparitor causes $H = 0 \rightarrow 1$. The inputs E and G to the AND gate prevent the possibility of $H = 1$ during reset or the beginning of forward t

when the comparitor input is within the comparitor dead space. The input to L on the comparitor locks $H = 1$ until $I = 0 \rightarrow 1$ causes $G = 1 \rightarrow 0$. This assures $H = 1$ even when the integrator is held so long that it drifts out of the comparitor dead space. H is an important logic output because it indicates when $\gamma(\eta)$ is available.

The determination of $t = t_s$ is made by a comparitor with input $f(t, \eta)$ and output J. The programming is straightforward and is shown in block I of Figure 7. A switch and comparitor connected to $v(t)$ generate $|v(t)|$. Since $v(t)$ is obtained from the adjoint equations the integrator for $f(t, \eta)$ is controlled by F. J is 0 until $t = t_s(\eta)$.

The rest of Figure 7 shows the programming of (13) and master control of the iterative cycle. On the completion of a run η^j is updated. This computation is shown in block II. Before the computer begins the first run $S = 0$. This establishes the initial condition on the memory units, η^o. At the end of the first run $S = 1$ and $H = 0 \rightarrow 1$ which causes η^1 to replace η^o. The inputs E and I for Figure 6 are derived from the logic elements in Figure 7. When $S = 0$, $I = 0$ and the γ program is reset. When $S = 0 \rightarrow 1$ the first run begins. Since $f(0, \eta^o) > 0$, $J = 0$ and $E = 0 \rightarrow 1$. Thus the γ program begins. When $t = t_s(\eta^o) = t^o_s$, $E = 1 \rightarrow 0$. The L input on the comparitor holds E at 0 even though the f integrator is reset by $F = E$. At $2t^o_s$, $H = 0 \rightarrow 1$, indicating that $\gamma(\eta^o)$ is available, triggers the memory units. At $2t^o_s + \triangle$ the single shot causes $I = 0 \rightarrow 1$. This resets the γ program causing H and J to return to 0. The delay $\triangle$ guarantees that $H = 1$ long enough to trigger the single shot. At $2t^o_s + \triangle + 1$ the single shot returns I to 0 beginning the second run.

The choice of ρ is important. If it is too large t_s^{j+1} may be smaller than t_s^j, indicating that an overstep in the correction of η^j has been made. Conversely, if ρ is too small the convergence of the η^j to η^* may be very slow. It would be helpful in the selection of ρ to have an overstep indicator in the computer program.

Figure 8 shows such an indicator. The memory unit is triggered by J and therefore stores the value of t_s on the previous run. If during the present run t_s is larger, the comparitor produces a l output at the end of the forward t portion of the run. The first AND gate with input E assures that this comparison is examined only during forward t. The first flip-flop is cleared by I at the beginning of each run. If an overstep has not been made it will be set during the run by the comparitor. Clearly, if the first flip-flop is set, the second flip-flop will remain cleared since H cannot pass through the AND gate. If however there is an overstep, the second flip-flop will be set by H and $K = 0 \rightarrow 1$ thus indicating an overstep.

An equipment count in Figure 5 through Figure 8 yields the following requirements:

Integrators	$2n + 2$
Switches	$n + 4$ (2 single throw)
Comparitors	5
Track-transfers	$n + 1$
AND Gates	9
Flip-flops	3
Single-Shots	1

CONCLUSIONS

A hybrid computer of the type described appears to be well suited for the solution of certain optimal control problems. The required flexible control of analog elements is obtained easily with a reasonable number of logic elements. There seems to be no real obstacle in extending the techniques outlined above to more complex control problems and iterative cycles, and eventually, to on-line control computation for actual plants.

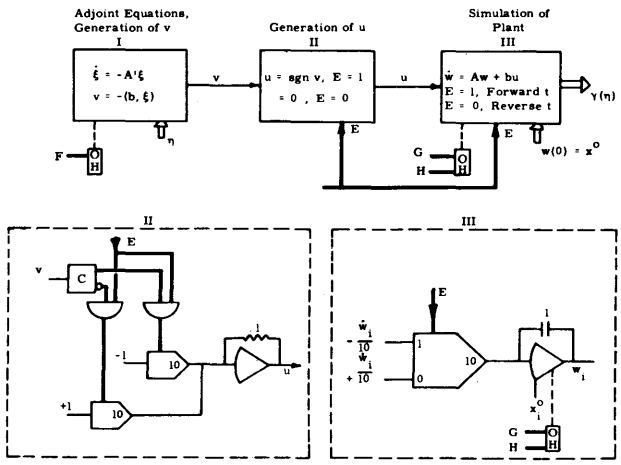


Figure 5. Computer Program for Generation of $\gamma(\eta)$.

ACKNOWLEDEMENT

The author thanks Edward O. Gilbert and Edward J. Fadden for helpful discussions on hybrid computing.

REFERENCES

1. J. P. LaSalle, "The Time Optimal Control Problem," *Contributions to Theory of Nonlinear Oscillations*, vol. 5, Princeton Univ. Press. 1960.
2. L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*, Interscience Publishers, New York, 1962.
3. L. W. Neustadt, "Synthesizing Time Optimal Control Systems," *Journal of Math. Analysis and Applications*, vol. 1, pp. 484-493, 1960.
4. E. G. Gilbert and E. J. Fadden, article to appear.

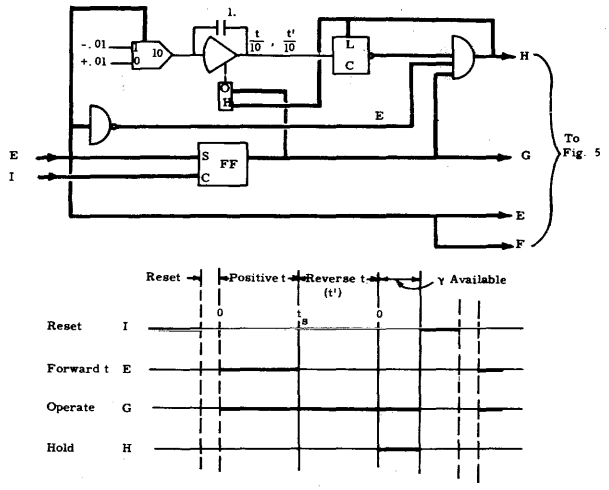


Figure 6. Control Program for Figure 5.

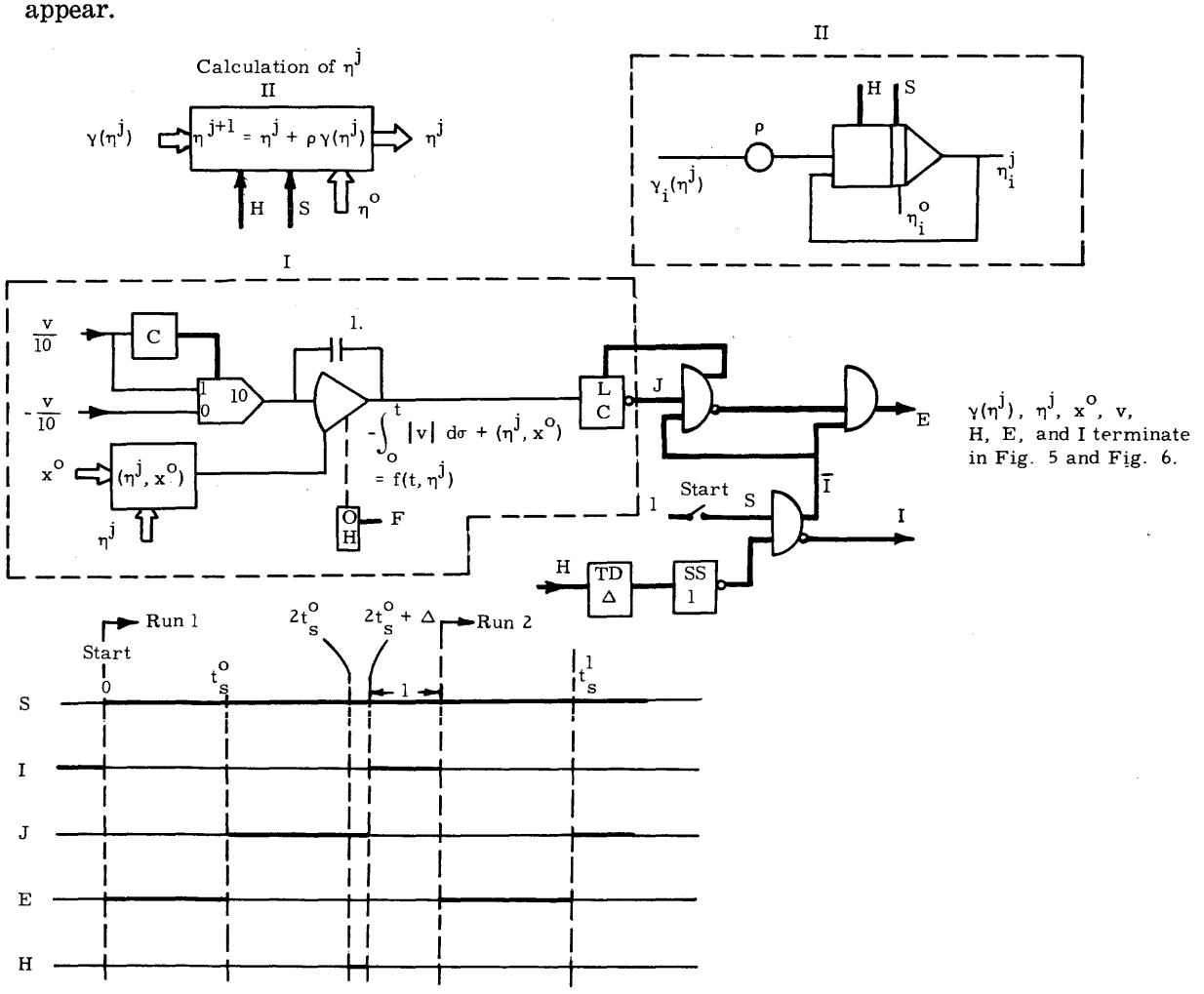


Figure 7. Iterative Solution of the Optimum Regulator Problem.

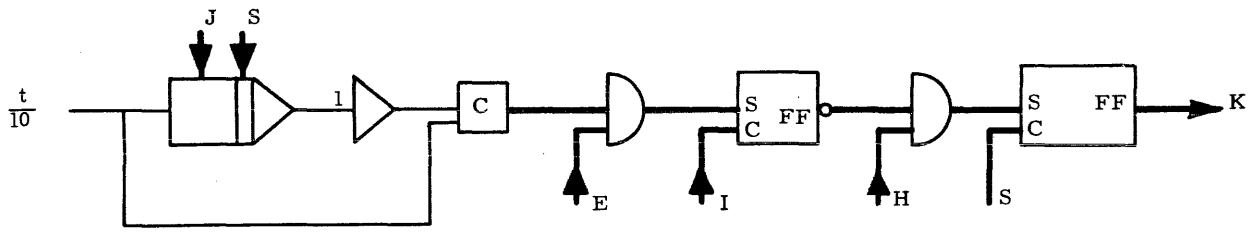


Figure 8. Program for Overstep Determination.

MULTIPLE INTEGRALS ON A NON-REPETITIVE ANALOG COMPUTER

Arthur Hausner
United States Army Materiel Command
HARRY DIAMOND LABORATORIES
formerly
Diamond Ordnance Fuze Laboratories
Washington 25, D. C.

INTRODUCTION

Multi-variable problems are among the most difficult types of problems for an electronic analog computer to solve, for such a computer is a single-independent-variable machine. Approximations are invariably made and the mathematics used is frequently borrowed from the domain of numerical analysis. The area of multiple integrals is no exception. A typical numerical technique for evaluating an integral of the type

$$I = \int_a^b \int_{y_1(x)}^{y_2(x)} f(x, y)\, dy\, dx \qquad (1)$$

is to divide the interval from a to b into n intervals $\triangle x_i$; as in Figure 1. Then

$$I \approx I_A = \sum_{i=1}^{n} \Delta x_i \int_{y_1(x_i)}^{y_2(x_i)} f(x_i, y)\, dy \qquad (2)$$

where x_i is the mean value of x in the interval $\triangle x_i$. It is well known that $I_A \to I$ when $n \to \infty$ and $\max(\triangle x_i) \to 0$. Generally, it is easiest to compute the approximation of I by taking $\triangle x_i$ constant. Then, $\triangle x_i = \triangle x$, and

$$n\Delta x = |a - b|. \qquad (3)$$

For a sufficiently large n, a reasonable approximation is obtained. To minimize n, various quadrature formulas[1] are available. Rubin, Laudauer, and Totten[2] used this technique with a 16-point Gaussian quadrature formula in the evaluation of a set of six double integrals involved in antenna pattern calculations on an analog computer. This method affords a reasonable compromise between accuracy and time of computation, as well as reducing the amount of equipment required. (All functions of x are constants during the calculation in each interval.)

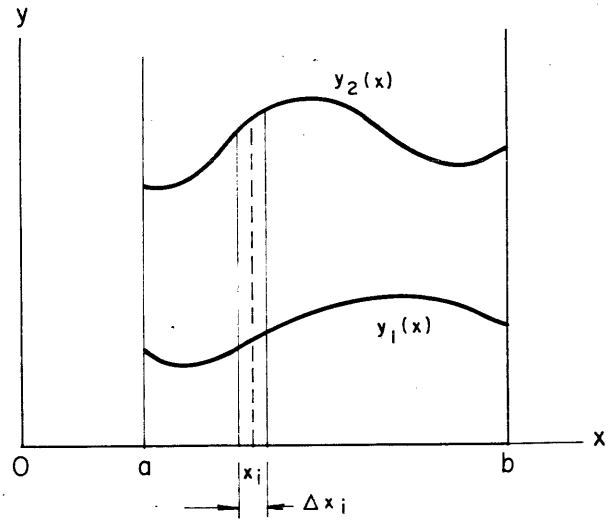


Figure 1. Interval from a to b Divided into n Intervals of Δx_i Width.

The above method is essentially a digital one, and would become tedious on an analog com-

puter for integrals of order higher than two. Rogers and Connolly[3] suggest the use of a scanning technique for higher-order integrals. Each variable is forced to change continually in a sawtooth fashion. No details are given in their book.

This paper develops techniques for solving (1) that allow x to vary continually from a to b as y oscillates quickly between y_2 and y_1. The scan is a continuous path and affords an automatic and convenient method of analog computation. The formulas derived do not require the path to be linear, although such a path is by far the most convenient. Both diode and relay circuits are given, as well as two illustrative examples of how the circuits are used. Extensions to higher-order integrals are also made, the accuracy of computation diminishing as the order of the integral increases.

When a repetitive computer with memory is used, a suggested method for calculating multiple integrals[4] introduces an error due to mathematical approximation, even in limiting cases such as $\frac{\partial f}{\partial y} = 0$. The techniques used here introduce no approximation error for such a situation. One easily concludes that the methods introduced here are preferable even when a repetitive computer with memory is available.

MATHEMATICAL APPROXIMATIONS

By distorting the path of integration to allow for a continually increasing x, as in Figure 2, the following approximation to I holds:

$$I \approx I_A = \sum_{i=1}^{n} \Delta x_i \int_{y_1(x_i)}^{y_2(x_{i+1})} f(x, y)\ |dy| \qquad (4)$$

where $\triangle x_i = x_{i+1} - x_i$, and x in this interval is a single-valued function of y but not necessarily linear. The integration is performed with respect to $|dy|$ to allow for a descending path from y_2 to y_1, without changing the sign. (We assume $y_2(x) \geq y_1(x)$). It follows that as max $(\triangle x_i) \to 0$ and $n \to \infty$, then $I_A \to I$.

For the purposes of analog computation, y is made a single-valued function of x and is forced to oscillate between $y_2(x)$ and $y_1(x)$. Mathematically, we seek a y such that

$$y = (y_2 - y_1)\,\Omega\,(x) + y_1 \qquad (5)$$

where $\Omega(x)$ is a continuous function of x that oscillates between 0 and 1, with bounded derivatives. $\Omega(a)$ can be either 0 or 1, and it is best that $\Omega(b)$ be either 0 or 1, to eliminate any partial intervals. Then

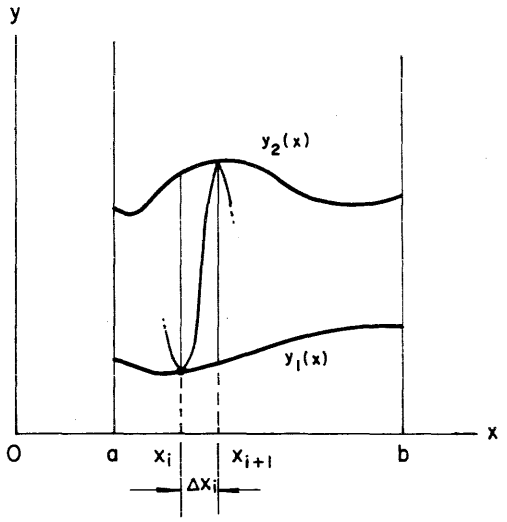


Figure 2. Distorting the Path of Integration to Provide Continuity.

$$\frac{dy}{dx} = \frac{dy_1}{dx} + \left[\frac{dy_2}{dx} - \frac{dy_1}{dx}\right]\Omega(x) + (y_2 - y_1)\frac{d\Omega(x)}{dx}. \qquad (6)$$

The frequency of oscillation is required to be high so that the dominant term of (6) will be $(y_2 - y_1)\ \dfrac{d\Omega(x)}{dx}$.

Hence

$$\frac{dy}{dx} \approx (y_2 - y_1)\frac{d\Omega(x)}{dx} = (y_2 - y_1)\Omega'(x) \qquad (7)$$

and

$$|dy| = (y_2 - y_1)|\Omega'(x)|dx. \qquad (8)$$

Equation (4) reduces to

$$I \approx \sum_{i=1}^{n} \Delta x_i \int_{x_i}^{x_{i+1}} f(x, y)(y_2 - y_1)\,|\Omega'(x)|\,dx. \qquad (9)$$

Equation (9) is the basis for making I the result of a continuous integration of a single variable.

THE CHOICE OF $\Omega(x)$

Of the more common oscillatory patterns, the triangular wave offers the advantage of simplicity. Sinusoidal waves have been considered,[5] but introduce additional multiplications and

additional equipment. Their only advantage is to eliminate switching problems which occur when triangular waves are considered. Hence, we let

$$\Omega(x) = k(x - x_i) \text{ for } x_i \leq x \leq x_{i+1} \quad (10)$$
$$\text{and} \quad = 1 - k(x - x_{i+1}) \text{ for } x_{i+1} \leq x \leq x_{i+2}$$

for $k > 0$ and odd i. Since $x_1 = a$, we have chosen $\Omega(a) = 0$ without loss of generality. The function y for two successive intervals becomes

$$y = y_1 + k(y_2 - y_1)(x - x_i) \quad (11)$$
$$\text{and} \quad = y_2 - k(y_2 - y_1)(x - x_{i+1})$$

for odd i.

In each interval, $\Omega(x)$ changes by one unit (from 0 to 1 or 1 to 0), so that from (10)

$$1 = k\Delta x_i. \quad (12)$$

Since

$$|\Omega'(x)| = k, \quad (13)$$

$$I \approx \sum_{i=1}^{n} \frac{1}{k} \int_{x_i}^{x_{i+1}} kf(x, y)(y_2 - y_1)\, dx$$

$$\approx \int_a^b f(x, y)(y_2 - y_1)\, dx. \quad (14)$$

Note that the slope k has dropped out of (14) before summation so that the approximation holds if the area bounded by $a \leq x \leq b$ and $y_1 \leq y \leq y_2$ is traversed by straight lines of any slope. The slope k may be permitted to change slowly and vary from interval to interval, an important concept when consideration of generating (11) on the computer is made. Since $y_2 - y_1$ will be, after proper scaling, a slowly changing function of x, for two successive intervals (11) is amended to be

$$y = y_1 + c(x)(x - x_i) \quad (15)$$
$$\text{and} \quad = y_2 - c(x)(x - x_{i+1})$$

where $c(x)$ any positive function of x which is roughly constant in any interval*.

The equations to be solved, from (14) and (15) are

$$\frac{dI}{dx} = [y_2(x) - y_1(x)]f(x, y) \quad (16)$$

and

$$\frac{dy}{dx} = \pm c(x). \quad (17)$$

* This formulation accommodates the parametric technique for avoiding division[7] whenever denominators contain function of x.

In differentiating (14), b is replaced by x. It is understood that the computation ends when $x = b$. In differentiating (15) to obtain (17), it is assumed that y_1, y_2, and $c(x)$ are constants, or that

$$\frac{dy_1(x)}{dx}, \frac{dy_2(x)}{dx}, \text{ and } \frac{dc(x)}{dx}$$

are small compared to $c(x)$.

MECHANIZATION ON THE COMPUTER

Three distinct problems are present in the solution of (16) and (17) by an analog computer:

1. Generation of $y_2(x)$ and $y_1(x)$
2. Generation of y
3. Generation of $f(x,y)$

We shall be concerned only with the generation of y. The techniques and difficulties of generating functions of one or more variables are discussed in virtually every text on analog computation. This paper can only be applied if all functions required by (16) are generable.

The function $c(x)$ is virtually arbitrary and is selected for convenience. The problem involved in mechanizing (15) is to switch the sign of the input to the integrator producing y when y reaches either y_2 or y_1. One method of doing this is illustrated in Figure 3 with $c(x)$ proportional to $y_2(x) - y_1(x)$. The bang-bang circuit is strongly stable and diode-limited. That y remains between y_2 and y_1 can be observed by tracing the output of the high-gain amplifier when the summer output is both positive and negative. Figure 4 gives an equivalent circuit using a high-speed relay for switching. Note

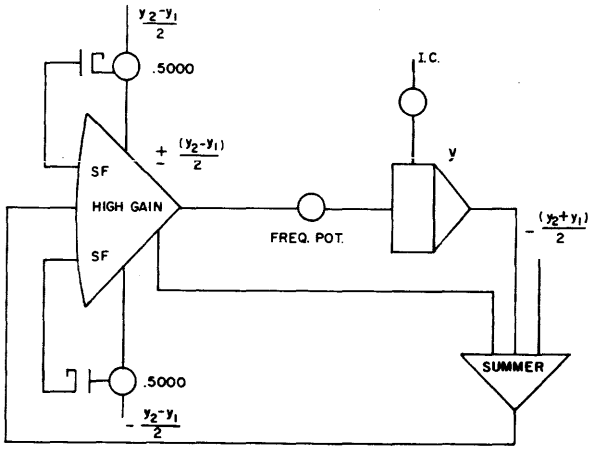


Figure 3. Diode Switching for $c(x)$ Proportional to $y_2 - y_1$.

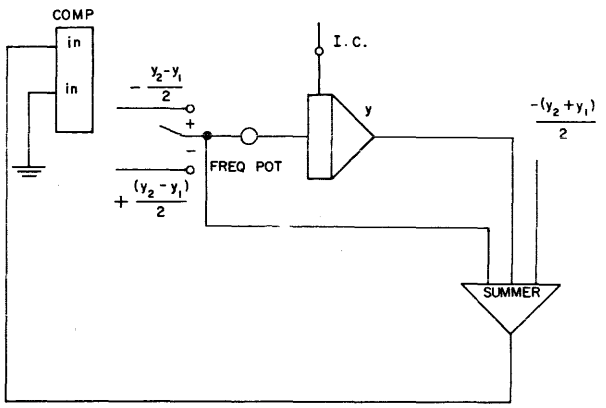


Figure 4. Relay Switching for $c(x)$ Proportional to $y_2 - y_1$.

that the frequency potentiometer in both cases serves only to change the number of intervals and has no bearing in the solution of (16) and (17). However, the approximation (4) is improved when n is large. It is desirable, therefore, to obtain as high an input rate as possible.

One obvious method of getting the largest possible number of intervals is to let $c(x)$ be constant, for it may then be scaled to 100 v. When diodes are used for switching in this case (fig. 5) more equipment is required than when relays are used (fig. 6). Which circuit to use is dependent upon the type of equipment available. If high speed double-pole double-throw switches are available, the circuit shown in figure 6 uses a minimum of amplifiers and undoubtedly provides the greatest accuracy.

The rest of the solution of (16) and (17) is straightforward. Since $y_2(x) - y_1(x)$ is a relatively slowly changing variable, it may be used to drive the shaft of a servo multiplier. The function $f(x,y)$ must be generated with high-speed equipment, as y will be a rapidly changing function of time.

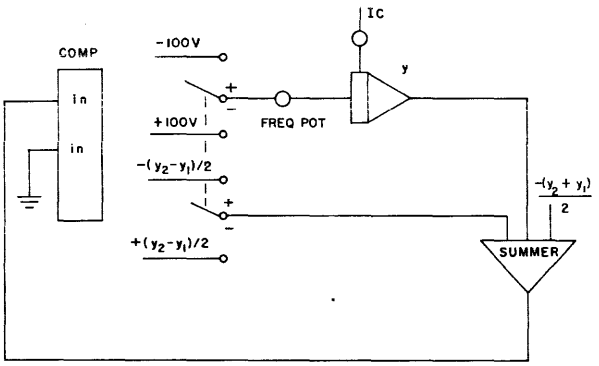


Figure 6. Relay Switching for Constant $c(x)$.

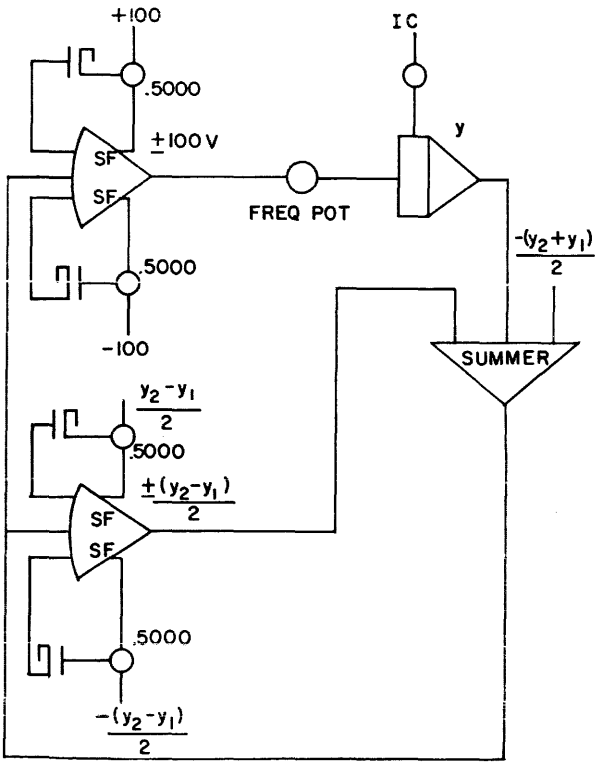


Figure 5. Diode Switching for Constant $c(x)$.

ERROR AND ACCURACY

It is necessary to distinguish between approximation error and computation error, even though they are not mutually exclusive. The limitation of the amplifiers in reproducing high frequencies limits the number of intervals into which the interval from a to b may be decomposed. An effort to decrease the approximation error by requiring a higher frequency of oscillation may very well decrease the net accuracy due to a substantial increase in the computation error. A case in point is the example in the next section where a change in the number of intervals from 30 to about 800 does not materially improve the net accuracy. One may generally assume that the approximation error is negligible (by analog computer standards) unless there are relatively high-frequency components in $f(x,y)$, y_2, or y_1.

No approximation error exists in this technique when the integrand of the double integral is a function of x only:

$$\int_a^b \int_{y_1(x)}^{y_2(x)} g(x)dy\,dx = \int_a^b g(x)\,[y_2(x) - y_1(x)]\,dx. \quad (18)$$

The approximation expressed by (14) is then exact. One would not use multiple integral techniques for such a situation, but this example serves well to show qualitatively why the techniques used here are preferable to those suggested when a repetitive computer with memory is available. The basic scheme suggested[4] is to compute, store, and up-date the inner integral

$$\int_{y_1}^{y_2} f(x, y)dy$$

at a high repetitive rate as x is continually increasing. The output of the memory is a staircase function of x and is integrated to obtain the required double integral. Even in the trivial case given in (18), one is required to use a high repetitive rate in order to minimize the approximation error. This is due only to the staircase function which appears at the output of the memory. One may conclude generally that techniques involving memory introduce more approximation error for an evaluation of a multiple integral than the techniques introduced in this paper.

It was determined experimentally that computation error increased greatly when changes in y exceeded 5000 v/sec for all examples tried*. This upper limit would be greater if the frequency responses of the equipment were improved. Besides normal computing errors, non-ideal diodes and relays contribute to the computation error. For a 5000 v/sec excursion for y, 100 v is attained in 20ms. Relays are inapplicable unless switching times are in the microsecond ranges. Simple diode circuits, on the other hand, will not give a constant output. Not only does y deviate from its linear relationship with x, but also the switching occurs at values other than y_2 and y_1. The use of idealized diode circuits, however, completely eliminate these errors although they require more equipment. Langill's multivibrator circuit[6] may be modified to satisfy the requirements on y and produce accurate diode switching.

The accuracy with which all the test double integrals were evaluated was better than one percent, with many solutions giving better than 0.1 percent accuracy, when using the simple diode circuits shown in this paper.

* An Electronics Associates, Inc., PACE 131-R Analog Computer was used. Tests were made with simple diode limiting circuits because no fast-acting relays were available.

EXAMPLE

For demonstration, we take

$$I = \int_{-1}^{1}\int_{0}^{\sqrt{1-x^2}} 2\pi y dy dx \qquad (19)$$

which represents the volume of a sphere of radius 1. By (14) we use

$$I \approx 2\pi\int_{-1}^{1} y\sqrt{1-x^2}dx = 4\pi\int_{0}^{1} y\sqrt{1-x^2}dx \qquad (20)$$

with the understanding that y is generated by any one of the circuits shown in figs. 3-6. A computer diagram, shown in Figure 7, utilizes a servo multiplier for finding the square root and for multiplication. Note that the output of the integrator representing I is in reality

$$I(x) \approx 4\pi\int_{0}^{x} y\sqrt{1-x^2}dx. \qquad (21)$$

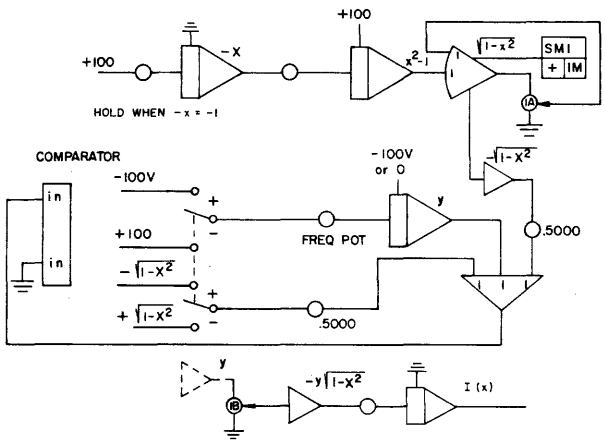


Figure 7. Flow Diagram for Solution of

$$I = 4\pi\int_{0}^{1}\int_{0}^{\sqrt{1-x^2}} y\, dy\, dx.$$

With an automatic hold circuit, the computation is stopped when $x = 1$, and the value of I is determined. Note also, the initial condition on y could be either 0 or 100 v. A good procedure is to make both runs and to average the value of I obtained.

Figure 8 shows a typical graph of $y(x)$ obtained with the relay circuit as shown in figure 7. The number of intervals is comparatively small, and the computation time was made 100 seconds so as not to exceed the dynamic range of the variplotter in order to obtain the graph. With a gain of 1, the frequency potentiometer was set at 0.2, resulting in slopes of ±20 and

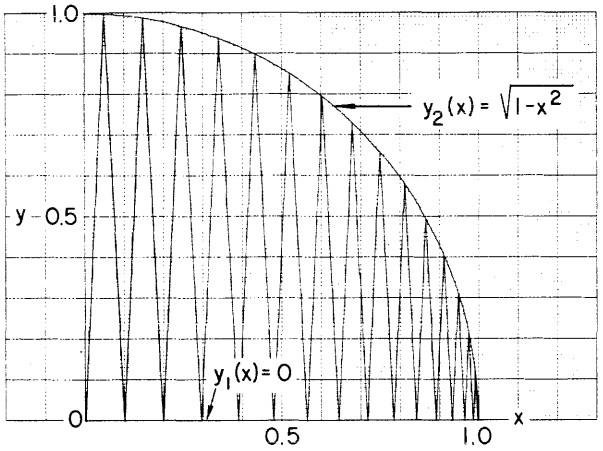


Figure 8. A Typical $y(x)$ Curve With Slopes of ± 20.

guaranteeing a minimum of 20 intervals*. Note that the size of the intervals decreases as y_2 approaches y_1 whenever the slopes are constant, as in this case.

Figure 9 shows the graph of $I(x)$ obtained. Note that the curve oscillates about the true answer which is superimposed on the graph:

$$I(x) = 4\pi\int_0^x \int_0^{\sqrt{1-x^2}} y\,dy\,dx = 2\pi\left[x - \frac{x^3}{3}\right]. \quad (22)$$

The results are most accurate at the end of each interval.

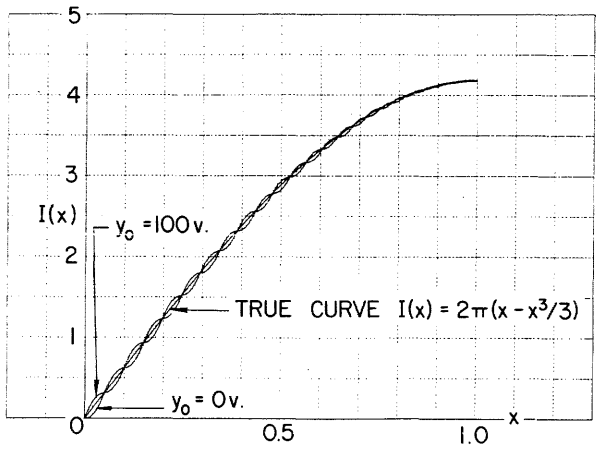


Figure 9. $I(x)$ Approximation to $4\pi\int_0^x \int_0^{\sqrt{1-x^2}} y\,dy\,dx$ Using Slopes of ± 20.

* When the slopes are constant, the number of intervals n obtained can be shown to be close to

$$n = s\int_a^b \frac{dx}{y_2 - y_1}$$

where s is the slope. For this example, $n = \pi s/2$ which equals 31 when $s = 20$. By actual count from figure 8, there are 30 intervals.

In order to increase the slopes and shorten computation time, the diode circuit for producing y as shown in figure 5 was used next. The computation time was shortened to 10 seconds, and 5000 v/sec was obtained for y by using a $0.1\mu F$ feedback capacitor with a $0.2M\Omega$ input resistor. This produced slopes of ± 500 and about 800 intervals. Because the oscillations are small, $I(x)$ as shown in Figure 10, appears as a relatively smooth curve.

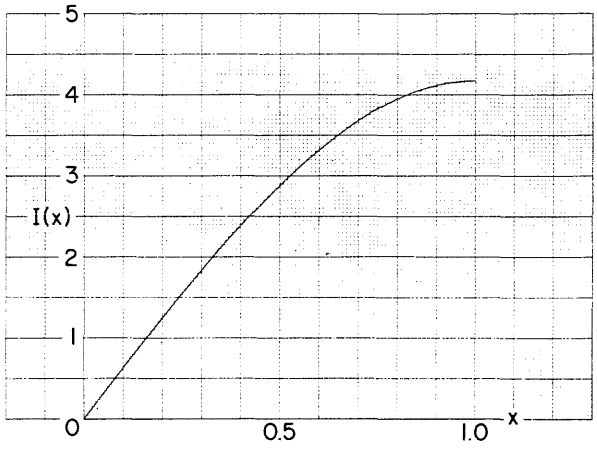


Figure 10. $I(x)$ Approximation to $4\pi\int_0^x \int_0^{\sqrt{1-x^2}} y\,dy\,dx$ Using Slopes of ± 500.

EXTENSIONS TO HIGHER-ORDER INTEGRALS

Integrals of the form

$$I = \int_a^b \int_{y_1(x)}^{y_2(x)} \int_{z_1(x,y)}^{z_2(x,y)} f(x, y, z)\,dz\,dy\,dx \quad (23)$$

are amenable to analysis similar to that which has been applied to integrals of type (1). Formulas similar to those of (14) are obtained if straight line paths are used:

$$I \approx \int_a^b (y_2 - y_1)(z_2 - z_1) f(x, y, z)\,dx \quad (24)$$

where

- z oscillates between z_2 and z_1, at the highest frequency;
- y oscillates between y_2 and y_1, at a medium frequency; and
- x traverses from a to b.

In general, one requires n-1 different frequencies of oscillation for an n^{th}-order integral. The slowest variable does not oscillate but traverses from one end of its limit to the other. The accuracy of the computation is probably great-

est if the ratio of successive frequencies of oscillation are equal. Thus the rate at which the variables change should satisfy a geometric progression. If the slowest variable is moved at p v/sec and the fastest variable at q v/sec, the ratio r of successive voltage rates satisfies

$$q = pr^{n-1}. \tag{25}$$

The i^{th} intermediate voltage rate R_i is then given by

$$R_i = pr^i = p\left\{ q/p \right\}^{\frac{i}{n-1}}. \tag{26}$$

For the PACE 131-R Analog Computer used in this study, a third-order integral would have voltage rates of 1, 70, and 5000 v/sec. A typical flow diagram for the solution of (23) is shown in Figure 11. One may decrease p to increase the ratio of voltage rates, but integrator drift rates prevent too low a limit. If the independent variable traverses 100 v during the computation, then the computing time is 100 sec. Decreasing the computing time is accomplished with a sacrifice of accuracy. A change of voltage rates to 5, 160, and 5000 v/sec decreases the computing time to 20 sec for a third-order integral, but cuts the number of volume elements by the same factor of 5, and the approximation error increases. Fifth-order test integrals have been evaluated with reasonable success with voltage rates of 0.5, 5, 50, 500, and 5000 v/sec. In general, one can expect the accuracy to diminish quite severely for higher-order integrals unless sufficiently large voltage ratios are obtainable. With the same computer, approximate accuracy limits for a few sample integrals were 1, 2, 5, and 10 percent for second-, third-, fourth-, and fifth-order integrals. Usually, the results were better, but this was highly dependent on the problem.

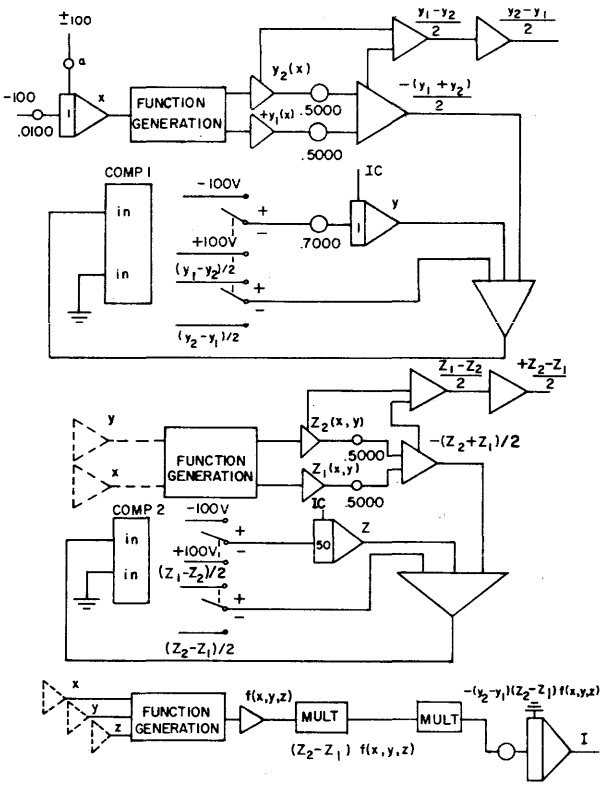


Figure 11. Flow Diagram for a Third-Order Integral.

ANOTHER APPLICATION

If a simple integral contains a parameter x

$$I(x) = \int_{a(x)}^{b(x)} f(x, y)dy \tag{27}$$

then

$$\frac{dI}{dx} = \int_{a(x)}^{b(x)} \frac{\partial f(x, y)}{\partial x} dy + f(x, b)\frac{db}{dx} - f(x, a)\frac{da}{dx} \tag{28}$$

from which

$$I(x) = \int_{x_o}^{x}\int_{a(x)}^{b(x)} \frac{\partial f(x, y)}{\partial x} dydx + \int_{x_o}^{x}\left[f(x, b)\frac{db}{dx} - f(x, a)\frac{da}{dx}\right]dx + I(x_o)\cdot \tag{29}$$

To apply this technique $I(x_o)$ must be known or calculated for $x = x_o$ for use as an initial condition. In addition, the derivatives as indicated in (29) must be generable. The procedure is not unlike the generalized integration method.

When the frequency of oscillation of y is large, the amplifier output which represents the double integral appears as a relatively smooth curve, so that (27) is reasonably represented by (29).

For example

$$I(x) = \int_0^x e^{-xy^2}dy \tag{30}$$

yields

$$I'(x) = \int_0^x - y^2e^{-xy^2}dy + e^{-x^3} \tag{31}$$

and

$$I(x) = \int_0^x\int_0^x - y^2e^{-xy^2}dydx + \int_0^x e^{-x^3}dx\cdot \tag{32}$$

Figure 12 shows a graph of (30) as approximated by changing (32) to

$$I(x) \approx \int_0^x \left[e^{-x^3} - xy^2 e^{-xy^3} \right] dx \qquad (33)$$

and generating y with the diode circuit of figure 5 with slopes of ±500. The term xy^2 was formed by an electronic multiplier, e^{-xy^3} by a diode function generator and e^{-x^3} by a generalized integration process.

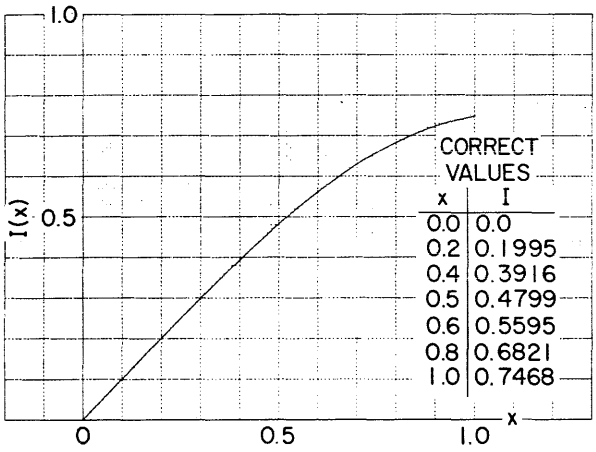


Figure 12. Double Integral Approximation to $I(x) = \int_0^x e^{-xy^2} dy$.

CONCLUSIONS

Multiple integrals are easily evaluated with non-repetitive analog computers and components generally available therein. The approximation formulas derived are inherently more accurate than those used with techniques involving memory. No approximation error is accumulated if $f(x,y)$ in (1) is equal to 1, and the integral represents the area bounded by a, b, $y_2(x)$ and $y_1(x)$. In fact, (14) is exact if $f(x,y)$ does not contain y. Therefore these techniques are preferable even for repetitive computers with memory.

ACKNOWLEDGEMENT

The author would like to accredit Dr. E. W. Chittenden (formerly of Harry Diamond Laboratories) with the basic idea of using a continuous path of integration which provided the stimulus for this paper.

REFERENCES

1. KOPAL, Z., "Numerical Analysis," John Wiley & Sons, Inc., New York, N.Y., p. 347, 1955.
2. RUBIN, A. I., LAUDAUER, J. P., and TOTTEN, H. Q, "Far Field Antenna Pattern Calculations by Means of a General Purpose Analog Computer", *Proceedings of the National Electronics Conference,* Volume XV, Oct., 1959.
3. ROGERS, A. E. and CONNOLLY, T. W., "Analog Computation in Engineering Design," McGraw-Hill Book Co., Inc., New York, New York, pp. 254-255, 1960.
4. Application Data Sheet 0012, Computer Systems, Inc., Monmouth Junction, New Jersey.
5. HAUSNER, A., "Multiple Integrations on a Real-Time Analog Computer," Diamond Ordnance Fuze Laboratories, Washington 25, D. C., Technical Report-1048, 30 July 1962.
6. LANGILL, JR, A. W., Accurate Simulation of Nine Common Non-Linearities", *Electronic Design,* p. 38, 21 June 1961.
7. HAUSNER, A., "Parametric Techniques for Eliminating Division and Treating Singularities in Computer Solutions of Ordinary Differential Equations", *IRE Transactions on Electronic Computers,* p. 42, February 1962.

HYBRID TECHNIQUES FOR ANALOG FUNCTION GENERATION

W. E. Chapelle
The Bendix Corporation
Research Laboratories Division
Southfield, Michigan

INTRODUCTION

In computing systems involving real-time simulation, data processing, or control, it is often possible to combine digital and analog computing techniques to improve the over-all system capability or performance. Effective combinations of analog and digital techniques have been made at several levels. At the systems level, general-purpose digital and analog machines have been linked through conversion equipment, enabling more effective solution of classes of problems which involve both precision arithmetic operations and rapid solution of differential equations. At the component level the capabilities of digital equipment for information storage, logical decisions, and sequence generation can be used in a variety of ways to augment the capabilities of analog equipment for wideband computation. Finally, at the information representation level, hybrid codes have been utilized[1, 2] to represent variables to obtain high-accuracy computing structures from an assembly of less accurate components.

This paper discusses techniques for employing hybrid concepts at the component and information representation levels to generate arbitrary functions. Particular emphasis is placed on the generation of multivariant functions in analog systems. One reason for this emphasis is that while currently available diode function generators are adequate for univariant function generation, they generally result in a prohibitive amount of circuitry in multivariant applications. Approaches involving servos have been used to reduce the equipment requirements, but these techniques are limited in the areas of speed and programming flexibility. A multivariant function generation capability can, of course, be obtained by incorporating a digital computer in the system. The present approach, however, obtains this capability at a considerably lower level of complexity, is generally capable of higher speeds, and does not contain the limitations of sampled data systems.

In general terms, generation of an arbitrary function requires two basic operations:

(1) *Storage*—Values of the output function are stored, the number of values depending on the required functional accuracy. In analog systems, the function values are normally stored on potentiometers, often implicitly in the form of slope values. If the function is not defined at predetermined points (i.e., fixed breakpoints), it is also necessary to store the values of the input variables at which the breakpoints occur.

(2) *Interpolation*—After the function is defined in terms of stored values, the output is formed by interpolating among these values as a function of the input variables. In most analog systems, first order interpolation is used.

The capabilities of digital techniques for information storage are well known. These techniques, of course, are most advantageous where the amount of stored data becomes large. This

condition tends to occur in multivariant function generation. For example, airframe simulations often involve the generation of ten or more bivariant functions, each of which may require storage of several hundred function values and breakpoint locations. Thus, the total storage requirements are in the order of several thousand quantities. Similar requirements occur in the generation of just one function of three variables.

The major objective of the following paragraphs is to show how the interpolation process can be effectively implemented by employing hybrid data representation codes and utilizing a combination of digital and analog circuits. It will be shown that these techniques are useful even in cases where storage requirements more modest than those cited above imply that potentiometer storage will be more economical than digital storage techniques.

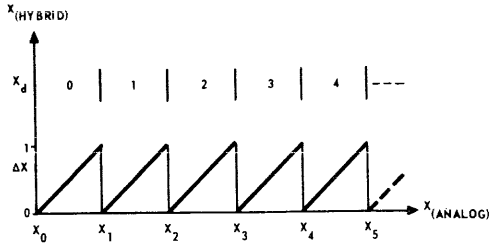
As noted above, the major emphasis herein will be on techniques for multivariant function generation. However, many of the concepts involved apply equally well to the single-variable case, and are more readily visualized at this level. Hence, the procedure in the following paragraphs will be to show first how hybrid techniques can be applied in the univariant case, and then to extend these techniques to functions of more than one variable.

BASIC CONCEPTS

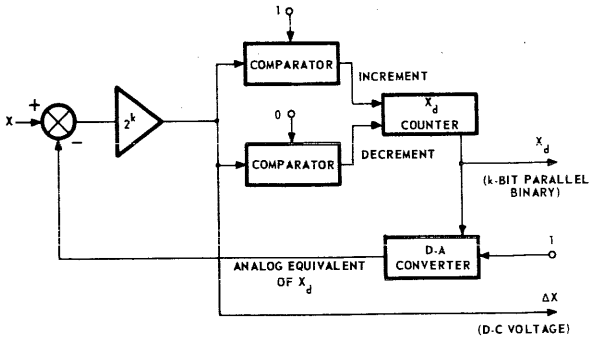
In a conventional hybrid code, a variable, X, is represented by the sum of two quantities: a digital number, X_d; and a d-c analog voltage, ΔX. For the present discussion, X_d can be visualized as a short binary number, say 3 or 4 bits, which would be the most significant bits of X in a normal binary representation. As indicated in Figure 1a, each value of X_d corresponds to a particular segment within the range of X:

$$X_d = j \text{ for } X_j < X < X_{j+1}$$

Within each segment, the analog quantity, ΔX, varies from zero to a maximum value which represents one bit of X_d. In practice, ΔX is scaled to vary over the full range of the machine variable, which is assumed here to be zero to unity, in order to utilize the full dynamic range of the equipment. This scaling leads to



(a) - Variation of X_d and ΔX as a Function of X



(b) - Analog-To-Hybrid Converter

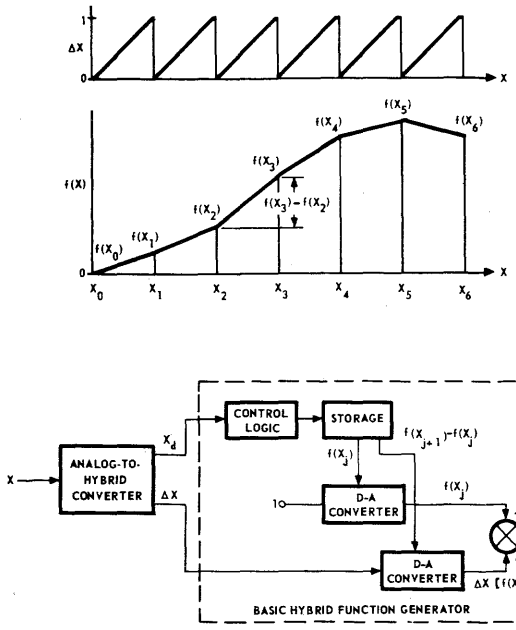
a definition for ΔX in a normalized form, which is convenient for the following discussion:

$$\Delta X = \frac{X - X_j}{X_{j+1} - X_j} \tag{1}$$

Equation 1 defines ΔX as varying linearly from 0 to 1 within each segment, independent of the width of the segment in X.

The conversion of X from a d-c analog voltage to the hybrid code is obtained with a circuit similar to the familiar closed-loop analog-to-digital converter, as shown in Figure 1b. The term X_d is generated as a k-bit parallel binary number, which is converted to analog and differenced with X. This difference is scaled to form ΔX in a precision gain section. The amplifier gain of 2^k is based on 0 to 1 full scale ranges for X, ΔX, and the analog equivalent of X_d. Comparators increment the X_d counter when ΔX exceeds unity and decrement X_d when ΔX becomes smaller than zero.

Schmid[3] describes a linear-segment analog function generator which utilizes a hybrid code generated in the above manner. This circuit, shown in Figure 2, provides a good starting point for the present discussion. The output function, $f_o(X)$, is defined at fixed evenly-spaced breakpoints located at each of the transition points of X_d. Each segment of $f_o(X)$, then, is permanently associated with a particular



value of X_d. For each segment, the storage section is programmed with two quantities:

(1) $f(X_j)$, the value of the function at the j^{th} breakpoint.

(2) $[f(X_{j+1}) - f(X_j)]$, the first difference of the function values for the interval $X_j < X < X_{j+1}$.

The value of X_d is used to control the readout of the stored function and first difference values into two parallel D-A converters which, respectively, have unity and ΔX as analog inputs. (The D-A converters can be visualized as the standard resistor-matrix parallel decoder, with the analog input replacing the normal reference voltage.) The D-A converter outputs, which are the products of their analog and digital inputs, are summed to form

$$f_0(X) = f(X_j) + \Delta X[f(X_{j+1}) - f(X_j)], \quad (2)$$

which is a linear segment approximation of the desired function.

In Schmid's function generator, the control logic consists of a decoder with 2^k output lines, one of which is energized for each segment. The storage section is a diode matrix, driven by the decoder, which can generate an independent parallel number for each function value and first difference. The program is stored by the presence or absence of a diode at each bit location. Figure 2 has been drawn in the more general form to emphasize that any form of digital memory can be utilized, provided that the access time is sufficiently short. It is also possible to modify Figure 2 for potentiometer storage; however, this concept, for convenience, is introduced in a later section.

The configuration shown in Figure 2 has two characteristics which are disadvantageous, particularly when the approach is expanded to the multivariant case:

(1) The fixed, evenly spaced breakpoint locations limit the programming flexibility. A movable breakpoint capability is desirable for functions which change slope rapidly in some places and are straight in others. It is almost essential when data are given in tabular form.

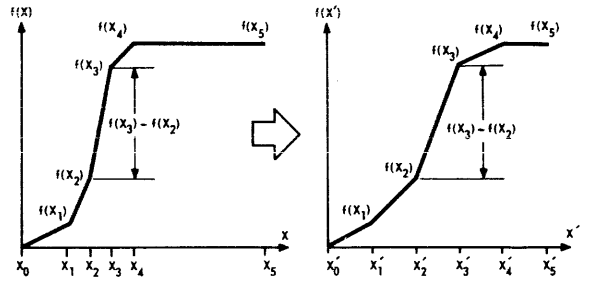
(2) The requirement for storing first differences, in addition to the function values, represents a redundancy which increases both the programming effort and the storage requirements.

The following paragraphs develop techniques for overcoming these disadvantages.

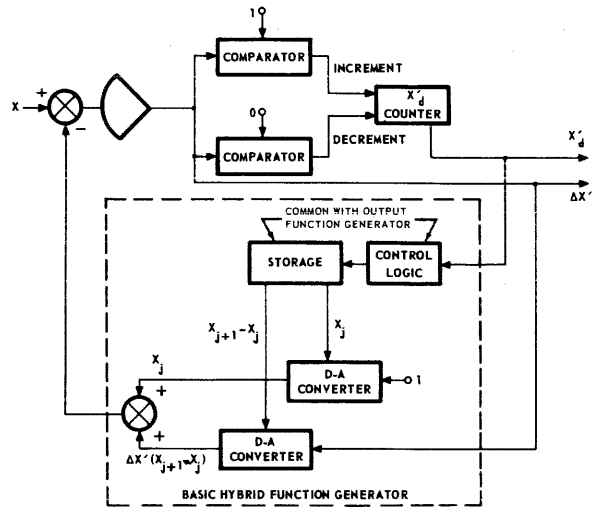
MOVABLE BREAKPOINTS

Consider a function $f(X)$ which is defined at non-uniform intervals in X as shown in Figure 3a. Now suppose this function is re-plotted on a new axis, X', on which the breakpoints are evenly spaced. Note that the function values and their first differences remain unchanged in this process. Hence, if a function generator of the type shown in Figure 2 is programmed with $f(X_j)$ and $[f(X_{j+1} - f(X_j)]$ and excited with the new variable X' the correct function will be generated.

It remains, then, to implement the mapping of the independent variable from X to X'. This can be done with another function generator; however, a much more efficient method is to modify the analog-to-hybrid (A-H) conversion circuit already present in Figure 2 to a function generator which forms X' (Hybrid) from X (Analog). In Figure 3b, the required nonlinear conversion is obtained by placing the basic hybrid function generator, developed in Figure 2, in the feedback path of a closed-loop system and replacing the programmed function values and first differences with the X-breakpoint locations and their first differences. The output of this function generator is differenced with X at the input of a high-gain amplifier, the



(a) - Univariant Mapping Relations



(b) - Mapping Configuration for Generation of Conventional Code

output of which is $\Delta X'$. Thus, within each segment, the closed loop system continuously solves the equation

$$X = X_j + \Delta X'(X_{j+1} - X_j) \qquad (3)$$

where $\Delta X'$ is the dependent variable. If Equation 3 is rewritten as

$$\Delta X' = \frac{X - X_j}{X_{j+1} - X_j}$$

it is clear that $\Delta X'$ varies linearly from 0 to 1 within each segment. The generation of X'_d, the digital part of X', is carried out in the same manner as before. Since X' is generated using a feedback loop, there must be no slope reversals or points of zero slope in the nonlinear conversion from X' to X. This is equivalent to requiring that the X breakpoints be assigned in order, $X_1 < X_2 < X_3 \cdots$.

Comparing Equations 1 and 3 it is seen that $\Delta X = \Delta X'$. This is true because both quantities are normalized to vary linearly from 0 to 1 within the segment. The reader may argue, then, that the introduction of the mapping concept is entirely academic—that Figure 3b is simply a circuit which generates the ΔX quantity required to implement Equation 2 in the presence of non-uniformly spaced breakpoints. This is true, of course; but the mapping concept aids in the visualization of a process which can become quite involved in the multivariant case.

Note that the X-breakpoint locations and their first differences are inserted in numerical form in the same manner as the function values and first differences. This is an important feature from the standpoint of programming ease.

DIRECT PROGRAMMING

In considering methods for eliminating the requirement for storing first differences, it is helpful to re-write Equation 2 in the form:

$$f_0(X) = (1 - \Delta X)f(X_j) + \Delta X f(X_{j+1}) \qquad (4)$$

Equation 4 expresses the output function within a given segment as a weighted mean of two stored breakpoint values, with the weighting determined by the instantaneous value of the independent variable, X, via the coefficients $(1 - \Delta X)$ and ΔX.

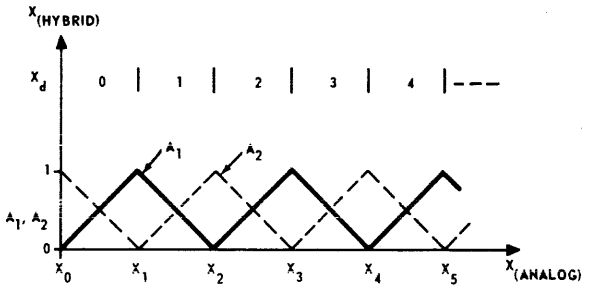
The implementation of Equation 4 involves the generation of a new hybrid code, which is plotted versus X in Figure 4. In the analog portion of the code, ΔX is replaced by two quantities, A_1 and A_2, which are defined as:

$$\left.\begin{aligned} A_1 &= 1 - \Delta X \\ A_2 &= \Delta X \end{aligned}\right\} \text{For } X_d \text{ Odd}$$

and

$$\left.\begin{aligned} A_1 &= \Delta X \\ A_2 &= 1 - \Delta X \end{aligned}\right\} \text{For } X_d \text{ Even}$$

Note that A_1 and A_2 are complementary; i.e., $A_2 = 1 - A_1$ and $A_1 = 1 - A_2$.



The quantities A_1 and A_2 excite the D-A converter configuration, shown in Figure 5, in which all the function values stored for odd numbered breakpoints (X_1,X_3,X_5, ...) are permanently associated with one D-A converter and those for even numbered breakpoints are permanently associated with the other D-A converter. (For convenience, in Figure 5 and future diagrams, D-A converters are shown as boxes marked with their digital inputs which are assumed to be generated by appropriate storage and control logic equipment.) Then, if the D-A converters are programmed according to the following schedule it can be seen that Equation 4 is obtained in each interval:

X_d	A_1	A_2	f_{odd}	f_{even}
0	ΔX	$1 - \Delta X$	$f(X_1)$	$f(X_0)$
1	$1 - \Delta X$	ΔX	$f(X_1)$	$f(X_2)$
2	ΔX	$1 - \Delta X$	$f(X_3)$	$f(X_2)$
3	$1 - \Delta X$	ΔX	$f(X_3)$	$f(X_4)$
....	....	....	....	....

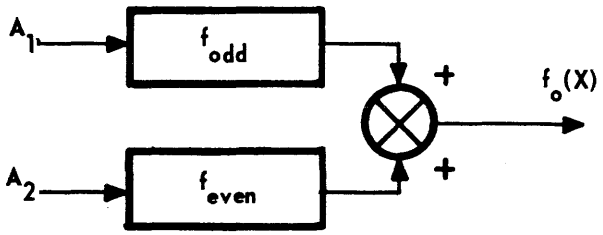
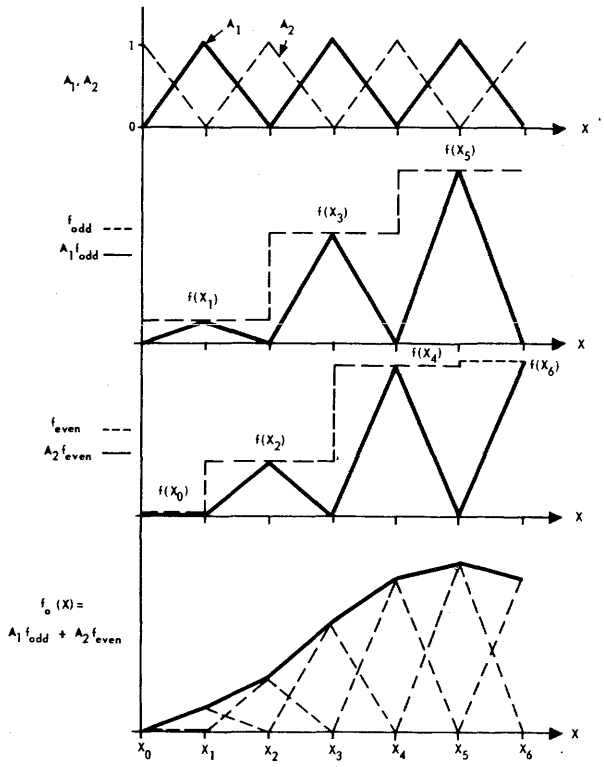


Figure 6 provides a graphical explanation of the synthesis of an arbitrary function using the odd-even code which makes the process easier to visualize. Within the first two segments, A_1 is multiplied by $f(X_1)$ to form a triangular function having height $f(X_1)$ as shown on line b. At X_2,$f(X_1)$ is replaced in the f_{odd} D-A converter by $f(X_3)$, so that the next triangle has a height of $f(X_3)$, etc. A_2 is multiplied by the even numbered function values to form the function shown on line c. The sum of the odd and even contributions (line d) is the desired function.

The configuration shown in Figure 5 is not a unique method for implementing Equation 4. However, the odd-even approach has several virtues which would not be present if the conventional code were utilized:

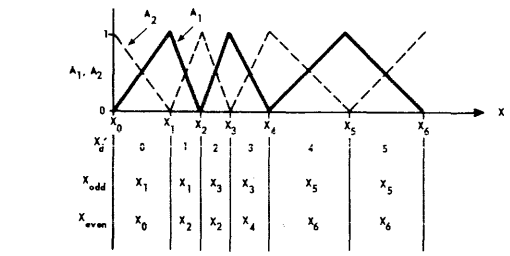
(1) The permanent association of each function value with one of the D-A converters eliminates switching of the stored values between D-A converters. This type of switching would be particularly difficult in the simpler forms of storage such as the diode matrix.

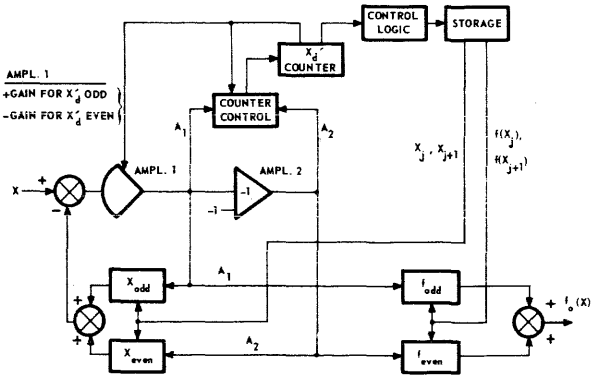


(2) All switching of function values in D-A converters occurs when the associated coefficient, A_1 or A_2, is equal to zero. This feature greatly reduces switching noise seen at the output of the function generator.

(3) The odd-even code, while more difficult to generate than the conventional code, contains no step changes and can therefore be used by analog equipment with less distortion due to limited amplifier high frequency response.

It remains to be shown how the odd-even hybrid code can be generated from a continuous analog input. This explanation will be carried out in terms of the previously discussed need for movable breakpoints, so that the result will be a movable-breakpoint univariant function generator which can be programmed directly with the function values and their locations in X. As shown in Figure 7, two odd-even D-A





converter configurations are connected back-to-back and driven from A_1 and A_2 which are, respectively, the outputs of Amplifier 1 which has high gain and Amplifier 2 which is connected to generate $A_2 = 1 - A_1$. Figure 7 also indicates the programming schedule for the D-A converters in the feedback path. The feedback loop solves the equation

$$X = A_1 X_{odd} + A_2 X_{even}$$

where A_1 is the dependent variable and $A_2 = 1 - A_1$.

In the first segment where X'_d is even

$$X = A_1X_1 + (1 - A_1)X_0$$

or

$$A_1 = \frac{X - X_0}{X_1 - X_0} = \Delta X \text{ and } A_2 = 1 - \Delta X.$$

In the next segment where X'_d is odd

$$X = A_1X_1 + (1 - A_1)X_2$$

or

$$A_1 = \frac{X_2 - X}{X_2 - X_1} = 1 - \Delta X, \text{ and } A_2 = \Delta X.$$

Note that the feedback path of the A-H conversion loop contains a gain term $X_{odd} - X_{even}$. Remembering that $X_0 < X_1 < X_2 \ldots$ it is seen that this term is positive during even numbered intervals and negative during odd numbered intervals. Thus, it is necessary to invert the sign of the gain of Amplifier 1 at each breakpoint. The information for this switching is contained in the least bit of X'_d.

The logic requirements presented in Figure 7 differ from those in Figure 2, in two ways. First, signals to increment or decrement X'_d, which are generated when either A_1 or A_2 become less than zero, must also be based on whether X'_d is odd or even:

	$A_1 < 0$	$A_2 < 0$
X'_d odd	Inc.	Dec.
X'_d even	Dec.	Inc.

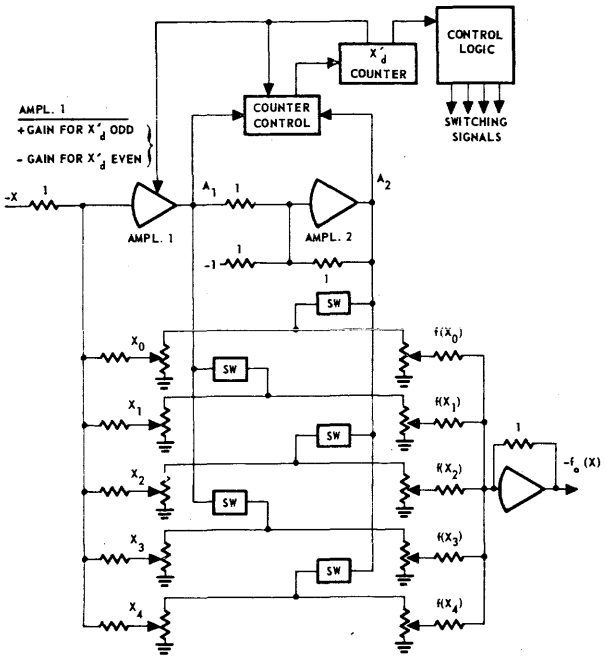
The second control logic modification occurs in the generation of the control signals for read-out of the stored quantities, where the following selection process is required.

f_{odd}, X_{odd}	X'_d	f_{even}, X_{even}	X'_d
$f(X_1), X_1$	0 or 1	$f(X_0), X_0$	0
$f(X_3), X_3$	2 or 3	$f(X_2), X_2$	1 or 2
$f(X_5), X_5$	4 or 5	$f(X_4), X_4$	3 or 4
• • • • • • • • •		• • • • • • • • •	

This selection can be implemented with a decoder for X'_d and one OR gate for each segment.

POTENTIOMETER STORAGE

The configuration shown in Figure 7 provides a convenient place to introduce the method, promised earlier, of using potentiometers to store the function values and breakpoint locations. Figure 8 shows a potentiometer-storage univariant function generator having the same capability as the Figure 7 configuration. Each function value and breakpoint location is stored as the shaft position of a separate potentiometer. Each potentiometer pair, X_j and $f(X_j)$, is connected to A_1 or A_2 by a switch during the $(j-1)^{th}$ and j^{th} segments, using exactly the same logic as for Figure 7. The potentiometer outputs, then, represent the products required, to implement Equation 4 and are summed to form the output function.



Any of the function generator block diagrams discussed in this paper can be converted to potentiometer storage by replacing the digital storage and the D-A converters with potentiometers and electronic switches. The required number of potentiometers is equal to the sum of the number of function values and breakpoint locations to be stored. The number of switches is equal to the number of function values.

BIVARIANT FUNCTION GENERATION

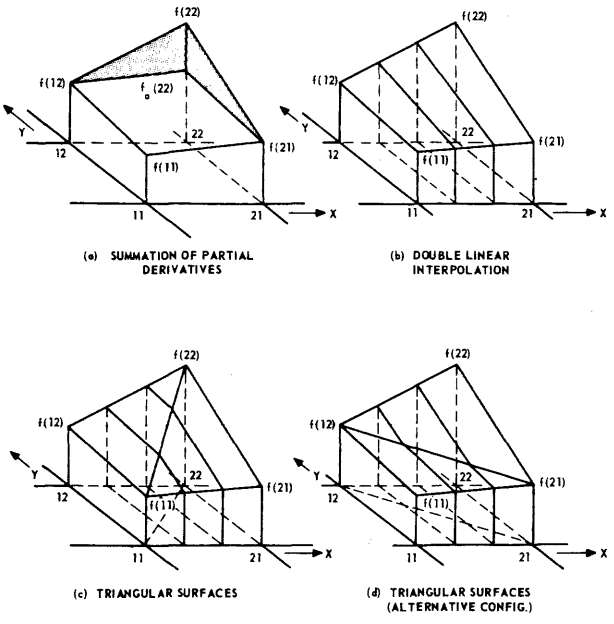
In considering the extension of the above concepts to the generation of functions of more than one variable, the first problem encountered is that the linear segment approximation for one variable has no unique counterpart in the multivariant case. Restricting the discussion to two variables for the present, Equation 2 (and its alternate form, Equation 4) can be expanded into at least three essentially different approximations to a surface $f(X, Y)$ which is defined by stored values at various points in the X-Y plane.

In the following paragraphs, these three approximations are briefly described, and their relative advantages are evaluated. Initially, it will be assumed that the function is defined in the X-Y plane at all the intersections of evenly spaced values of X and Y. (This does not provide a movable breakpoint capability, but this feature can be added as was done in the univariant case.) Thus, each approximation must define the function inside a square sector using stored values at the four corners. In general terms, the discussion should be carried out for the ji^{th} sector in which the function is defined by the corner values $f(X_j, Y_i)$; $f(X_{j+1}, Y_i)$; $f(X_j, Y_{i+1})$; and $f(X_{j+1}, Y_{i+1})$. However, this notation is much too complicated for the present requirements. It will suffice to denote the four values $f(11)$, $f(21)$, $f(12)$, and $f(22)$ as shown in Figure 9, and let it be understood that the square under discussion could be any sector in the X-Y plane.

The first approximation method, which might be called *summation of partial derivatives*, results from noting that Equation 2 gives $f(X)$ as the sum of a stored function value and a difference term which is proportional to the slope in the X-direction. A two-variable approximation, then, is obtained by adding a term which includes the effect of the slope in the Y-direction:

$$f(X, Y) = f(11) + \Delta X[f(21) - f(11)] + \Delta Y[f(12) - f(11)] \quad (5)$$

Equation 5 generates a plane surface which passes through $f(11)$, $f(21)$ and $f(12)$ as shown in Figure 9a. In general, the surface does not pass through $f(22)$—a plane defined by three points will not pass through an arbitrarily located fourth point. Thus, if the approximation is repeated in adjacent sectors, a discontinuity



will occur at each sector boundary as indicated by the shaded vertical areas in Figure 9a. For many applications these discontinuities present a severe disadvantage.

The implementation of Equation 5 is a direct extension of Figure 2. A third channel, in which the Y-first difference term is multiplied by ΔY, is added to the basic hybrid function generator. However, although an analogy to the odd-even method can be applied, the configuration cannot be rearranged to eliminate switching of non-zero quantities. Since the other approximations to be discussed present neither this disadvantage nor the discontinuous output surface, it is concluded that the summation of partial derivatives method is the least suitable of the three methods for wideband analog function generation.

DOUBLE LINEAR INTERPOLATION

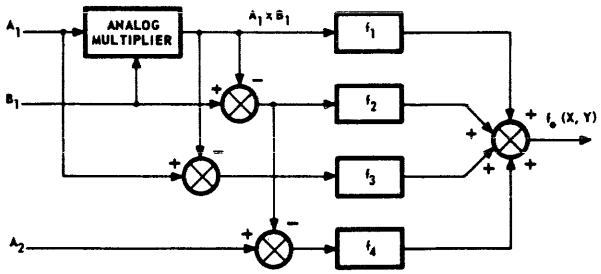
The second approximation, which will be called *double linear interpolation,* can be thought of as being formed by a linear interpolation in (say) the Y-direction between two functions of X which, in turn, are formed by linear interpolation between the corner values. Expanding the form of Equation 4 gives the following equation:

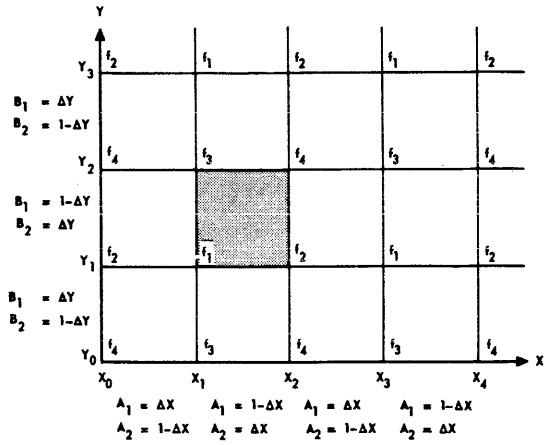
$$f_0(X, Y) = (1 - \Delta Y)[(1 - \Delta X)f(11) + \Delta X f(21)] + \Delta Y[(1 - \Delta X)f(12) + \Delta X f(22)] \quad (6)$$

Reversal of the order of the ΔX and ΔY interpolations will result in the same equation. The surface generated by Equation 6 is shown in Figure 9b. This is a curved surface which has the property of being straight along any line of constant X or constant Y.

Most approaches to bivariant function generation make use of this approximation by utilizing a linear interpolation device to interpolate between outputs of univariant function generators.[4, 5] This can be implemented in hybrid form, using two univariant circuits operating from a common control logic and storage section. In this approach, the linear interpolator requires the formation of two products of analog variables, and each multiplication must be performed with an accuracy equivalent to the desired output accuracy.

An alternate implementation which requires only one multiplication, and this of less accuracy, is shown in Figure 10. Figure 10 also extends the odd-even concept to the bivariant case. Each stored function value is permanently associated with one of four D-A converters according to the indicated schedule. Auxiliary signals are generated as follows:





$$\left.\begin{aligned} A_1 &= 1 - \Delta X \\ A_2 &= \Delta X \end{aligned}\right\} \text{For } X_d \text{ odd}$$

$$\left.\begin{aligned} A_1 &= \Delta X \\ A_2 &= 1 - \Delta X \end{aligned}\right\} \text{For } X_d \text{ even}$$

$$\left.\begin{aligned} B_1 &= 1 - \Delta Y \\ B_2 &= \Delta Y \end{aligned}\right\} \text{For } Y_d \text{ odd}$$

$$\left.\begin{aligned} B_1 &= \Delta Y \\ B_2 &= 1 - \Delta Y \end{aligned}\right\} \text{For } Y_d \text{ even}$$

The 1, 1 sector ($X_d = Y_d = 1$, shown shaded in Figure 10) is considered first. Here, Equation 6 becomes

$$f_0(X, Y) = (1 - \Delta Y)[(1 - \Delta X)f_1 + \Delta X f_2] + \Delta Y[(1 - \Delta X)f_3 + \Delta X f_4]$$

or

$$f_0(X, Y) = B_1(A_1 f_1 + A_2 f_2) + B_2(A_1 f_3 + A_2 f_4) \quad (7)$$

Moving up to the 1, 2 sector, Equation 6 becomes

$$f_0(X, Y) = (1 - \Delta Y)[(1 - \Delta X)f_3 + \Delta X f_4)] + \Delta Y[(1 - \Delta X)f_1 + \Delta X f_2];$$

and making the inverted substitutions, $1 - \Delta Y = B_2$ and $\Delta Y = B_1$, Equation 7 is again obtained. This equation also holds in the 2, 1 and 2, 2 sectors; and since all other sectors are repetitions of one of these four cases, Equation 7 is valid for the entire X-Y plane.

In Figure 10 the coefficients for all four function values are obtained with one analog multiplier by using the relations

$$A_2 = 1 - A_1 \text{ and } B_2 = 1 - B_1$$

to rearrange Equation 7 as follows:

$$f_0(X, Y) = A_1B_1f_1 + (B_1 - A_1B_1)f_2 + (A_1 - A_1B_1)f_3 + (A_2 - B_1 + A_1B_1)f_4. \quad (8)$$

In addition to the multiplier, four summing amplifiers are required to invert the multiplier output and obtain the required summations. Note that multiplier accuracy is generally not as important as it would be in an output interpolator because errors in the term A_1B_1 in Equation 8 tend to cancel. The error cancellation is greatest where the four stored values are nearly equal; i.e., where the slope of the function is small.

In a function generator having the configuration shown in Figure 10, the control logic section must perform the following type of selection for each of the four D-A converters:

f_1	X_d	Y_d
$f(X_1, Y_1)$	X_0 or X_1	Y_0 or Y_1
$f(X_3, Y_1)$	X_2 or X_3	Y_0 or Y_1
$f(X_1, Y_3)$	X_0 or X_1	Y_2 or Y_3
$f(X_3, Y_3)$	X_2 or X_3	Y_2 or Y_3

For a function being approximated by N segments in the X-direction and M segments in the Y-direction, the selection for all four D-A converters can be implemented with approximately $N + M$ OR gates and NM AND gates in addition to the X_d and Y_d decoders.

TRIANGULAR SURFACES

The third bivariant approximation, which will be called the *triangular surfaces approximation,* is based on passing plane surfaces through values of the function taken three at a time. For a unit square sector in the X-Y plane, the approximation consists of two triangular surfaces, as shown in Figure 9c. The equation of the line dividing the two triangles is $\Delta X = \Delta Y$. An equation for the right hand surface, where $\Delta X > \Delta Y$, is formed by adding two terms to $f(11)$ which account for the change in the function as the sector boundary is traversed from point (11) to (21) and then to (22):

$$f_0(X, Y) = f(11) + \Delta X[f(21) - f(11)] + \Delta Y[f(22) - f(21)] \text{ for } \Delta X > \Delta Y.$$

Similarly, for the left hand surface

$$f_0(X, Y) = f(11) + \Delta Y[(f(12) - f(11)] + \Delta X[f(22) - f(12)] \text{ for } \Delta Y > \Delta X.$$

Rearranging,

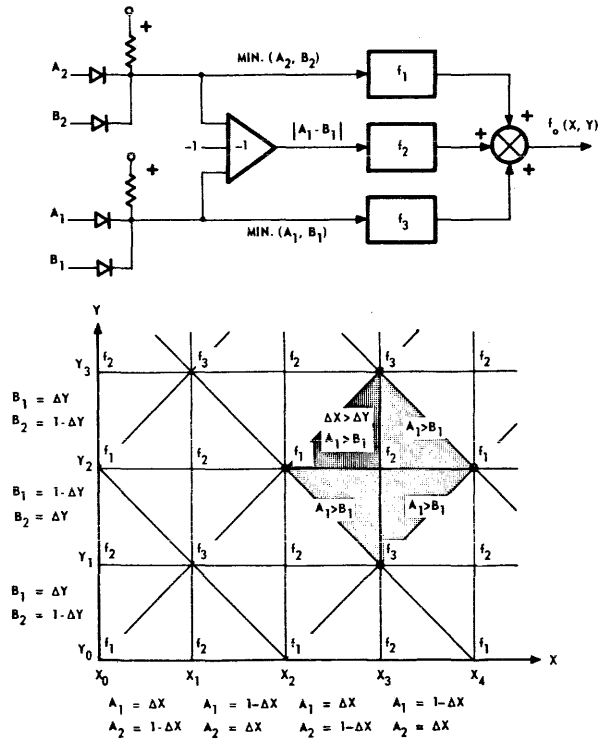
$$f_0(X, Y) = (1 - \Delta X)f(11) + (\Delta X - \Delta Y)f(21) + \Delta Y f(22) \text{ for } \Delta X > \Delta Y$$

and (9a)

$$f_0(X, Y) = (1 - \Delta Y)f(11) + (\Delta Y - \Delta X)f(12) + \Delta X f(22) \text{ for } \Delta Y > \Delta X. \quad (9b)$$

An alternate method for subdividing the unit square is shown in Figure 9d. In this case, the equation of the diagonal is $1 - \Delta X = \Delta Y$. Conceptually, these two methods are the same. Mathematically, one can be obtained from the other by complementing one of the input variables. However, it is interesting to note that the two methods do not, in general, yield the same functional error within a particular sector. Comparing these surfaces with Figure 9b, it is seen that the double linear interpolation method produces a surface which is, at all points, intermediate to the two possibilities available with triangular surfaces. Hence, it is deduced that the triangular surfaces method will generally produce a greater functional error, especially if the diagonal is located indiscriminately. On the other hand, it will be shown that the triangular surfaces approximation is considerably simpler from an equipment standpoint.

Figure 11 shows an implementation of Equation 9 which continues the concept of permanently associating each stored value with a particular D-A converter. The accompanying schedule indicates the assignment of function values to the three D-A converters and also



shows the diagonal locations, which have been selected to be compatible with the same set of auxiliary signals—A_1, A_2, B_1, B_2—that were obtained from ΔX and ΔY as in the previous example.

In showing how Figure 11 implements Equations 9a and 9b, the right hand portion of the 2, 2 sector (shown crosshatched) is investigated first. Here, $\Delta X > \Delta Y$; and by Equation 9a,

$$f_o(X, Y) = (1 - \Delta X)f_1 + (\Delta X - \Delta Y)f_2 + \Delta Y f_3$$

or

$$f_o(X, Y) = A_2 f_1 + (A_1 - B_1)f_2 + B_1 f_3 \quad \text{for } A_1 > B_1. \tag{10a}$$

In the triangle directly to the right of this, Equation 9a can again be utilized by complementing ΔX:

$$f_o(X, Y) = \Delta X f_1 + (1 - \Delta X - \Delta Y)f_2 + \Delta Y f_3$$

But, since moving across X_3 has reversed the definitions of A_1 and A_2,

$$f_o(X, Y) = A_2 f_1 + (A_1 - B_1)f_2 + B_1 f_3,$$

which is Equation 10a again. In a similar manner, or by applying Equation 9a, it can be shown that Equation 10a applies throughout the shaded region and in all other regions where $A_1 > B_1$.

In the rest of the regions, where $B_1 > A_1$, a similar development results in

$$f_o(X, Y) = B_2 f_1 + (B_1 - A_1)f_2 + A_1 f_3 \quad \text{for } B_1 > A_1. \tag{10b}$$

Equations 10a and 10b, taken together, define $f_o(X, Y)$ over the entire X-Y plane. However, for the purpose of discussing their implementation in Figure 11, it is desirable to combine Equations 10a and 10b into a common equation. This can be done by noting that $A_2 = 1 - A_1$ and $B_2 = 1 - B_1$, and observing the following:

(1) In Equation 10a, where $A_1 > B_1 : B_2 > A_2$ and $A_1 - B_1 > 0$
(2) In Equation 10b, where $B_1 > A_1 : A_2 > B_2$ and $B_1 - A_1 > 0$

Clearly, the coefficient of f_2 can be written $|A_1 - B_1|$ in both equations. The coefficient for f_1 can be written MIN (A_2, B_2), if MIN (A_2, B_2) is defined as being equal to A_2 or B_2 whichever is smaller. (This quantity could be called the analog AND of A_2 or B_2.) The coefficient of f_3 can be treated in the same manner. Thus,

$$f_o(X, Y) = [MIN(A_2, B_2)]f_1 + |A_1 - B_1|f_2 + [MIN(A_1, B_1)]f_3. \tag{11}$$

Figure 11 shows the implementation of Equation 11 using diode AND gates to form MIN (A_1, B_1) and MIN (A_2, B_2). The coefficient for f_2 is formed from

$$\begin{aligned} &1 - MIN(A_1, B_1) - MIN(A_2, B_2) \\ &= 1 - B_1 - 1 + A_1 = A_1 - B_1 \text{ for } A_1 > B_1 \\ &= 1 - A_1 - 1 + B_1 = B_1 - A_1 \text{ for } B_1 > A_1 \\ &= |A_1 - B_1| \end{aligned}$$

Thus, the required coefficients are obtained from A and B quantities with two AND gates and one amplifier. Diode AND gates can probably be utilized in most applications, inserting compensating diodes to cancel the first order effects of diode forward conduction offsets, and following the gate with a buffer amplifier. If higher accuracy is needed, each AND gate can be implemented with two D-C amplifiers. Either of these configurations compare favorably with the double linear interpolation approach from an equipment standpoint, since both eliminate the analog multiplier and one D-A converter.

The control logic required for selection of the f_1 and f_3 D-A converter inputs is the same as that for the double linear interpolation method. However, the selection for f_2 must also include the $A_1 > B_1$ or $B_1 > A_1$ condition:

f_2	A_1, B_1	X_d	Y_d
$f(X_1, Y_0)$	$A_1 > B_1$	X_0 or X_1	Y_0
$f(X_3, Y_0)$	$A_1 > B_1$	X_2 or X_3	Y_0
.....	.....	.....	.....
$f(X_0, Y_1)$	$B_1 > A_1$	X_0	Y_0 or Y_1
$f(X_2, Y_1)$	$B_1 > A_1$	X_1 or X_2	Y_0 or Y_1
.....	.....	.....	.....

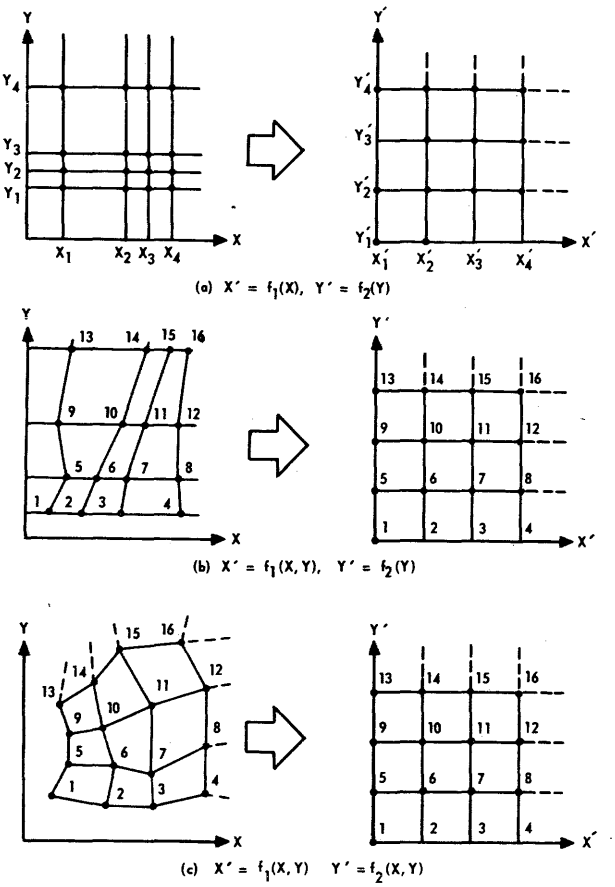
Thus, the control logic differs from that for the double linear interpolation method only in that a third input must be added to ½ NM AND gates.

In comparing the triangular surfaces approximation to the double linear interpolation approximation, the modest equipment savings available with triangular surfaces must be balanced against its generally higher functional error. No general conclusion can be reached on this point. However, it can be argued that stability and repeatability—which both methods provide in an approximately equal degree—are much more important than true functional accuracy in most analog applications. On this basis, it is concluded that the triangular surfaces approximation will generally be the optimum approach.

BIVARIANT MOVABLE BREAKPOINT SYSTEMS

If the approach for obtaining movable breakpoints that was developed for the univariant case is used separately for each input variable in a bivariant system, a mapping of the type shown in Figure 12a is obtained. Breakpoints located on arbitrarily spaced lines of constant X are translated to a uniform spacing in X', the only restriction being that order $X_1 < X_2 < X_3 \ldots$ be retained. The same type of translation is carried out for the Y breakpoints. The feedback paths of the X and Y analog-to-hybrid (A-H) converters employ the odd-even configuration, of course, so that the A_1, A_2, B_1, B_2 quantities required, in Figures 10 or 11, are generated.

In Figure 12b, the flexibility of the mapping has been increased by making the X breakpoints movable as a function of Y. This requires insertion of a bivariant function generator in the feedback path of the X-input A-H converter. Either of the bivariant configurations shown in Figures 10 or 11 can be utilized. The feedback path is programmed directly with



the breakpoint locations in X. The generation of the coefficients for the bivariant approximation can be common for both the output and X-feedback path.

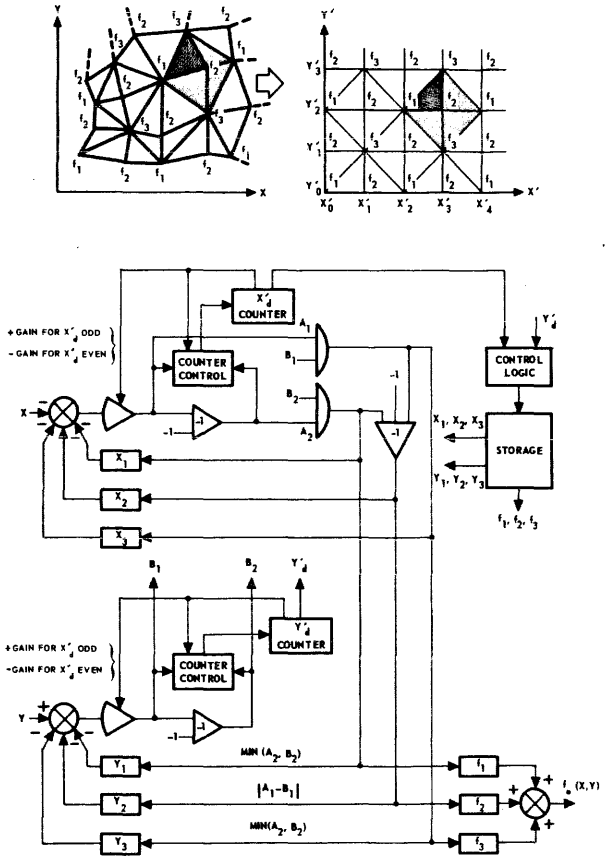
Figure 12c indicates a completely general bivariant mapping. Randomly located breakpoints in the X-Y plane are converted into a uniform grid in the X'-Y' plane. To obtain this mapping, a bivariant function generator is required in the feedback path of both the X- and Y-input A-H converters.

The three movable breakpoint systems then provide increasing programming flexibility along with increasing equipment complexity and, in particular, increasing storage requirements. For example, if a 10 x 10 segment approximation is implemented by each of the three methods, the storage requirements are:

	Number of Stored Quantities			
Mapping	X	Y	$f(X, Y)$	Total
12a	11	11	121	143
12b	121	11	121	253
12c	121	121	121	363

It may be argued that the intermediate mapping configuration, Figure 12b, is sufficient for most applications because raw data for bivariant functions is almost always presented as a set of functions of one variable with the other variable as the parameter of the set. However, since the configuration shown in Figure 12c is the most general case, it will be used in the following description of a complete bivariant function generator.

Figure 13 shows the implementation of a bivariant random-breakpoint analog function generator utilizing the triangular surfaces approximation. The output section (right half) of Figure 13 is essentially a repetition of Figure 11. The operation and logic requirements of the digital storage equipment are also as indicated in conjunction with Figure 11 except that three quantities—the value of the function, the X-coordinate, and the Y-coordinate—must be stored for each breakpoint. The assignment procedure for placing the X-coordinates in the X-input A-H converter and the Y-coordinates in the Y-input A-H converter is identical to that for placing the function in the output D-A converters.



The X- and Y-input A-H converters, then, simultaneously solve the equations

$$\left.\begin{aligned} X &= [MIN\ (A_2, B_2)]\ X_1 + |A_1 - B_1|\ X_2 + [MIN\ (A_1, B_1)]\ X_3 \\ Y &= [MIN\ (A_2, B_2)]\ Y_1 + |A_1 - B_1|\ Y_2 + [MIN\ (A_1, B_1)]\ Y_3 \end{aligned}\right\} \quad (12)$$

in which $\Delta X'$ and $\Delta Y'$, as contained in the A's and B's, are dependent variables.

A step-by-step proof that Equations 12 perform the required mapping is a lengthy procedure; consequently, only a summary is presented.

The first step is to show that each of the oblique triangles formed by joining breakpoints in the X-Y plane is mapped into a corresponding isosceles right triangle in the X'-Y' plane. This is done by showing that for any particular triangle Equation 12 is satisfied along all three boundaries, and that points in the interior of the triangle in X-Y remain in the interior in X'-Y'.

The second step is to show that the closed loop system formed by the X- and Y-analog-to-hybrid converters provides a stable solution of Equations 12. Classical stability criteria for implicit solution of simultaneous equations[6] can be applied to this problem to show that the following conditions, taken together, are sufficient for stability:

(1) An increase in X always corresponds with an increase in X'.
(2) An increase in Y always corresponds with an increase in Y'.
(3) The transformation does not invert the triangle; i.e., if the corners are numbered 1, 2, 3 clockwise in the X-Y plane, this order is not reversed in the X'-Y' plane.
(4) The phase shift in each amplifier is less than 90 degrees at all frequencies where gain exceeds unity.

The first three conditions restrict the choice of breakpoint locations in a manner analogous to the $X_1 > X_2 > X_3 \ldots$ requirement in the single variable case. Generally, they imply that two lines of constant X' (or Y') cannot cross in the X-Y plane and that no sector in X-Y can contain an interior angle of greater than 180 degrees.

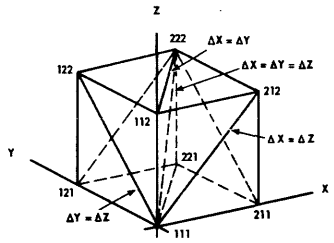
FUNCTIONS OF MORE THAN TWO VARIABLES

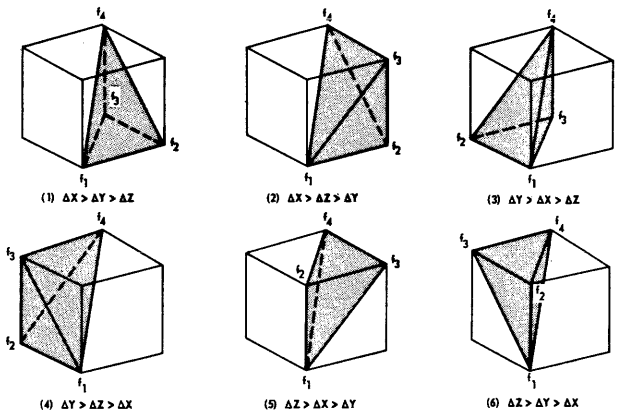
Of the three approximations discussed for bivariant interpolation, the first two have direct extensions for functions of three or more variables. For the summation partial derivatives method, a new first difference term is added to the basic equation, and another channel is added to the block diagram for each new variable. As in the bivariant case, the major disadvantage of this method is the discontinuous output.

A trivariant linear interpolation is formed by interpolating between the output of two bivariant function generators. Also, a trivariant extension of the configuration shown in Figure 10 is available. However, in either approach, a minimum of four analog multipliers are required.

The extension of the triangular surfaces approximation to functions of three or more variables is less direct. Generalizing, it can be argued that the virtue of the triangular surfaces approximation lies in the fact that the interpolation is carried out among the least number of points that can define a surface. Therefore, the trivariant analogy should implement an interpolation among the least number of points that can define a volume—four points in the form of a tetrahedron.

Figure 14 indicates a subdivision of unit cube into six tetrahedrons each of which is defined by a unique inequality among ΔX, ΔY, and ΔZ. Assuming the function is defined at the eight corners of the cube, and following the same procedure used in the bivariant case,





$$f_o(X, Y, Z) = f(111) + \Delta X[f(211) - f(111)] + \Delta Y[f(221) - f(211)] + \Delta Z[f(222) - f(221)] \quad \text{for } \Delta X > \Delta Y > \Delta Z.$$

Each of the four function values involved can be assigned to one of four D-A converters as indicated in the sketch of the $\Delta X > \Delta Y > \Delta Z$ tetrahedron in Figure 14. Thus,

$$f_o(X, Y, Z) = (1 - \Delta X)f_1 + (\Delta X - \Delta Y)f_2 + (\Delta Y - \Delta Z)f_3 + \Delta Z f_4 \quad \text{for } \Delta X > \Delta Y > \Delta Z.$$

Continuing, a set of six equations is obtained which defines the output function throughout the cube. Adopting the odd-even notation for the variables, the set becomes

$$\left.\begin{aligned} f_o(X, Y, Z) &= A_2f_1 + (A_1 - B_1)f_2 + (B_1 - C_1)f_3 + C_1f_4 \text{ for } A_1 > B_1 > C_1 \\ &= A_2f_1 + (A_1 - C_1)f_2 + (C_1 - B_1)f_3 + B_1f_4 \text{ for } A_1 > C_1 > B_1 \\ &= B_2f_1 + (B_1 - A_1)f_2 + (A_1 - C_1)f_3 + C_1f_4 \text{ for } B_1 > A_1 > C_1 \\ &= B_2f_1 + (B_1 - C_1)f_2 + (C_1 - A_1)f_3 + A_1f_4 \text{ for } B_1 > C_1 > A_1 \\ &= C_2f_1 + (C_1 - A_1)f_2 + (A_1 - B_1)f_3 + B_1f_4 \text{ for } C_1 > A_1 > B_1 \\ &= C_2f_1 + (C_1 - B_1)f_2 + (B_1 - A_1)f_3 + A_1f_4 \text{ for } C_1 > B_1 > A_1 \end{aligned}\right\} \quad (13)$$

The first and last term of Equations 13 can be generated as before with analog AND gates. The center terms, however, involve the intermediate quantity in the inequality and cannot be generated passively. They can, of course, be generated with active switching, employing the same logical process that must be implemented in the selection of the correct set of stored function values. The configuration retains the triangular surfaces' property of switching function values only when the coefficients are zero.

Since the tetrahedral volume approximation requires only four D-A converters and no analog multiplier, it possesses a considerable equipment advantage over the triple linear interpolation process. As more variables are added this advantage becomes greater. In a n-variable function generator, generalized interpolation requires $2^n - 2^{n-2} - 2$ multipliers and 2^nD-A converters; whereas, the generalized triangular surfaces approximation requires no multipliers and $n + 1$ D-A converters.

HARDWARE CONSIDERATIONS

In general, the equipment configurations that have been developed in the foregoing discussion can be implemented with an assemblage of standard components. The D-A converters, switches, and control logic circuits fall into this category. The amplifiers can be of the standard operational type with the one exception of the switched amplifier which is utilized in the generation of the odd-even hybrid code.

The switched amplifier, Amplifier 1 in Figure 7, provides the forward gain for a feedback loop whose feedback transfer function reverses sign at each segment boundary. To compensate for this reversal, the gain of the forward path must be inverted within Amplifier 1. The design should emphasize accomplishing this switching without generating a switching transient at the amplifier output, which is the signal voltage A_1. One rather conservative approach to this problem is to design Amplifier 1 as a short time-constant integrator with differential inputs, one for each feedback polarity, and then utilize the following switching sequence:

(1) When A_1 or A_2 less than zero is sensed, open the forward path placing Amplifier 1 on hold.
(2) Switch the feedback path by inserting the new value in the proper D-A converter.
(3) Close the forward path through the opposite input.

STORAGE METHODS

It has been noted that hybrid interpolation techniques are applicable with a variety of storage media. Each of these media, of course, has a characteristic set of advantages and limitations. The following paragraphs summarize some of these characteristics, for the major types of storage media.

Potentiometers

For requirements of less than about 1000 quantities, potentiometers probably are the most economical form of memory, assuming that manual setup procedures are employed. The disadvantage is that considerable labor is required in programming the equipment to generate a new function. Automatic potentiometer setting systems are in general more expensive than an equivalent digital memory.

Static Card Readers

The diode matrix memory proposed by Schmid can be increased in flexibility by making the connection at each bit location through the contacts of a static card reader. Readers are available which will handle about 1000 bits or (say) 100, 10-bit quantities. The disadvantages are a relatively high first cost and no reduction in cost per bit with increasing storage requirements.

Magnetic Cores

The fast random access capabilities of the core memory, together with the reducing cost per bit for large capacities, are ideal for large function generation capabilities. The only disadvantage is that, in order to justify the basic cost, the storage requirement must be in the order of 1000, 10-bit quantities; e.g., generation of several functions of two variables or one function of three variables.

Delay Lines, Drums

These storage media suffer generally from access time problems. If the full bandwidth capabilities of the hybrid function generation

approach are to be realized, access times of a few microseconds are required. Some improvement is available by inserting a buffer store to rapidly make available the values required in adjacent sectors.

The above discussion has emphasized storage methods which, by their programming flexibility, are best suited to general purpose computing applications. For special-purpose equipment, where a fixed or plug-in card program is suitable, the diode matrix storage is probably more economical than digital bulk storage methods up to a level of several thousand quantities.

CONCLUSIONS

Hybrid equipment and information representation techniques can be effectively applied to a wide class of analog function generation problems. The simpler configurations, such as shown in Figure 8, are comparable with the conventional diode function generators in the bandwidth and equipment complexity areas. They are advantageous in that the function values and breakpoint locations can be inserted directly from numerical data. Also, each function value and location is independent of all the others.

The major advantage of hybrid techniques, however, is that the above advantages can be extended to include generation of arbitrary functions of two or more variables with no great increase in the required equipment. An exception to this statement occurs in considering the storage requirements, where an equipment increase is inevitable. However, the hybrid approach makes efficient use of digital bulk storage techniques to handle the large amounts of data required to define multivariant functions.

In the discussion of bivariant approximation methods, it is concluded that the triangular surfaces approximation is optimum for wideband analog applications since it generates a continuous surface with the least amount of equipment. This argument is strengthened when the approximations are extended to functions of three or more variables. Differing system requirements may modify this conclusion, of course. For example, in systems which involve multiplexing of the variables, the continuous output has no great advantage, and therefore the summation of partial derivatives approximation may be adequate. In other cases, the fact that generalized linear interpolation is a more conventional approach mathematically or that it generally produces a lower functional error may justify the added equipment. Perhaps an additional advantage of hybrid function generation techniques is that they present a unified approach for implementing any of the three methods.

ACKNOWLEDGEMENT

The author acknowledges the contributions and many helpful comments of F. B. Lux and other staff members of the Information and Control Systems Laboratory at Bendix Research Laboratories during the investigation whose results have been described in this paper.

REFERENCES

1. SKRAMSTAD, H. K., "A Combined Analog-Digital Differential Analyzer," Proceedings of the Eastern Computer Conference, 1959.
2. SCHMID, H., "Combined Analog/Digital Computing Elements," Proceedings of the Western Joint Computer Conference, 1961.
3. SCHMID, H., "Linear-Segment Hybrid Function Generators," Proceedings of the Combined Analog Digital Computer Systems Symposium, December 16-17, 1960, Philadelphia.
4. CONNELLY, M. E., "Real-Time Analog Digital Computation," IRE Transactions on Electronic Computers, Vol. EC-11, No. 1, February, 1962.
5. SANSOM, J., "Function Generation," Instruments and Control Systems, Vol. 35, No. 1, pp. 127-129.
6. KORN and KORN, "Electronic Analog Computers," 2nd Edition, McGraw-Hill, 1956, pp. 64-66.

AUTOMATIC STRATIFICATION OF INFORMATION*

D. Lefkovitz and N. S. Prywes
The Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, Pa.

1. INTRODUCTION

Much of the development of science has been concerned with the organization of knowledge into strata. Within such a structure new developments are recorded and, as a result, may change the structural organization. Humans have found such stratified organizations useful for retrieving facts and for applying deductions to create what is a contemporary concept of civilization. In the field of information storage and retrieval we find similarly a need for structuring the information in strata. That is, the use of content addressed memories by themselves is not sufficient to solve the retrieval problem, and additional stratification of the descriptor language is necessary.

The stratification scheme proposed here consists of separation of a descriptor vocabulary, used to describe the items in a library, based upon two principles; namely, whether individual descriptors do or do not occur together in descriptions of items. This paper is directed specifically to the mechanization of stratification based on the latter of these principles. Thus the problem treated in the following is: Given an arbitrary information file how to arrange the descriptors of the data to constitute large groups of descriptors which are *exclusive* in that the descriptors in a group do not co-occur in the description of any item? For this purpose the input data are semi-automatically processed to form these "exclusive" attribute groups.

The proposed stratification scheme is flexible, so that changes in the structure of the descriptor language can be easily carried out. This is necessary for two reasons. (1) Every item input to the file, based on its description, is capable of changing the organization of the descriptor language. (2) We visualize a human-machine tandem system in which a human monitor can easily change the descriptor language organization, thereby affecting the processing of new items considerably. This might amount to a human starting or reorganizing periodically a filing system and letting the machine continue with the system. In this case, the human monitor teaches the machine by example.

The objectives of such an arrangement are several. One is the speed and storage-capacity efficiency in storage and retrieval of items as described further in Section 3. Another is to convey to the user semantic information regarding the sense given to descriptors used. These descriptors are usually words in a natural language which may normally be given a variety of meanings by a variety of people; each person using the words in a somewhat

* This work has been supported by the Information Systems Branch of the Office of Naval Research, under Contract NOnr 551(40). Use of the IBM 7090 Computer facility was provided through a grant from the Atomic Energy Commission.

restricted way.* Similarly the descriptors in the library must always be used to convey the desired meanings and not other unintended and unforeseen meanings. This may become more evident further through the discussion where the proposed stratification is intended to provide a structure for conveying to the user the "appropriate" meanings for natural language descriptor words. The aim is ultimately to mechanize information exchange between user and machine in phrasing retrieval requests and descriptorizing (indexing) new items (documents). The "appropriate" meanings of descriptors are defined only through association with other descriptors in descriptions of items (or documents) filed in the library. However, every new document may, if it is found by the indexer to be desirable and not confusing, extend the meaning of a descriptor in a natural language to convey additional concepts.

This paper however, due to space limitation, is directed to the problem of the stratification only. The processes described in this paper have been run on the IBM 7090. It is the intention of this paper only to introduce the problem and describe the solution strategy. A following paper will report in detail on the results of numerous experiments that have been performed to test the behavior of the stratification process when applied to various types of files.

The desirable stratified descriptor language is described in Section 2, and how this structuring fits into the retrieval system is discussed in Section 3. The problem that faces us after that is establishing the feasibility of a process leading to such a stratified structure. Several approaches to forming exclusive groups have been suggested; however, the entire vocabulary is involved in the process of accommodating changes.[1] Instead, to satisfy the flexibility needs stated above, a heuristic process is suggested, which progressively increases the number of descriptors involved until a desired change is accommodated. Toward this end a process is flow charted in Section 4 that, we judge, should achieve the desired stratification. Next, a small scale simulation has been carried out in Section 2 to further illustrate the process.

2. DESCRIPTION OF THE DESIRED STRUCTURE OF A DESCRIPTOR LANGUAGE

An entire file or library consists of many *items*. Each item is categorized by a set of *descriptor* words, called the *description* of the item. The totality of descriptors represents the *vocabulary* of the descriptor language, which is expandable.

The desired stratification of the descriptor language consists of separating the entire vocabulary into *attribute groups*. The descriptors in each attribute group represent *exclusive* values, i.e., no item description contains more than one descriptor from a single attribute group. Thereby an *attribute*-value stratification, illustrated in Figure 1, is obtained. The only condition here governing the separation of the descriptor vocabulary into attribute groups is that descriptors representing values in a single attribute would be exclusive. The attribute groups are defined as being *inclusive* in that a description of an item in the information file may contain descriptors belonging to any combination of attributes.

The field of information retrieval includes a diversity of files ranging over business, scientific and language data. The stratification of business data in the structure portrayed in Fig. 1 is sometimes obvious as its importance has been evident to the initiators of the particular

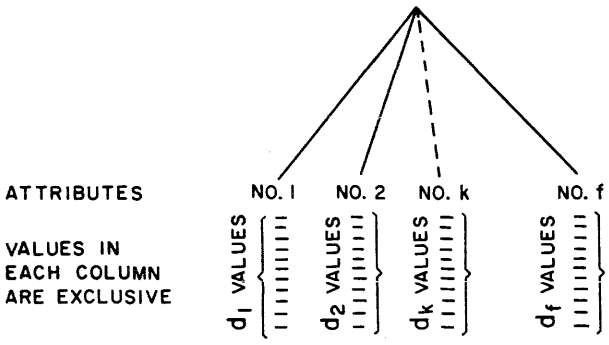


Figure 1. Schematic diagram of a descriptor language with two strata: f attributes and d_k exclusive values per attribute ($1 \leq k \leq f$).

* This is long known and well illustrated by the quotation: "When *I* use a word," Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean—neither more nor less."

"The question is," said Alice, "whether you *can* make words mean so many different things."

"The question is," said Humpty Dumpty, "which is to be master—that's all." *Through the Looking Glass*, Lewis Carrol, Macmillan and Co., London, 1872, p. 124.

business system. An example would be a military personnel file. Such a file may have a very large number of items but relatively few descriptors. The attributes in the descriptor language would be age, rank, serial number, name, height, etc., while the specific years of age, the specific rank, specific serial number, the alphabetic breakdown of the name, specific height, etc., would be values corresponding to the respective attributes. In scientific problems, the stratification of the descriptor language is very complex. It is the contention of this paper that both business and scientific problems can be organized into attribute-value strata and thereby handled efficiently in an information retrieval system.

The ASTIA Catalog is an example of a scientific information retrieval system. It differs from the personnel file in that it has a large number of descriptors. The structure of the ASTIA descriptor language is portrayed in Fig. 2. As is shown, it is organized in three strata: The descriptor vocabulary of up to 10,000 descriptors is divided among 19 fields which are distributed into 292 groups.* An example from this catalog is: Field: aeronautics, Group: aerodynamic configurations, Descriptors: air foil, airplane model, etc. The position of this example in the stratified ASTIA language is illustrated in the middle of Fig. 2. This stratified structure is generated by the human analysts. It is expandable by the analyst to include any degree of specialization but is lacking as to any organized or algorithmic structuring. Fig. 3 illustrates the structure of a descriptor language similar to ASTIA but structured in exclusive attribute-value groups. The stratification into three levels, similar to ASTIA can be maintained. The entire descriptor vocabulary is divided into f inclusive attribute-groups. On the average, then, the number of exclusive values per attribute-group is obtained by dividing the entire vocabulary (assumed to be 10,000) by the number of attribute-groups, f. (If f is taken as $f = 40$, then there are approximately 250 values per attribute; the choice of f will be discussed further.) The attribute-groups in Fig. 3 correspond in

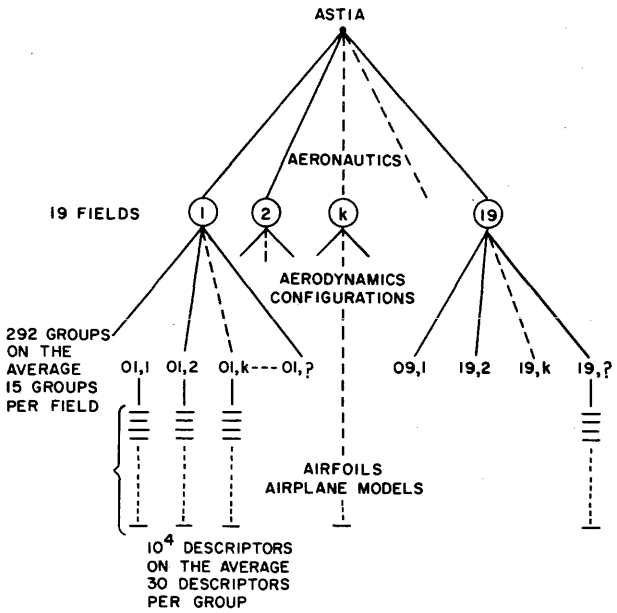


Figure 2. Schematic diagram of ASTIA descriptor language with three strata: 19 fields, 292 groups accommodating up to 10,000 descriptors.

strata level to the fields in Fig. 2. Another level of stratification is obtained by dividing the values corresponding to one attribute-group into the individual attributes which correspond to the same level as "groups" in the ASTIA system.**

It is our contention in the following that a descriptor language of the size of the ASTIA Catalog can be divided into a relatively small number of exclusive-attribute groups (less than 40). That such a separation into attribute groups is possible can be satisfactorily demonstrated in two ways. First, by experimenting with processes that perform such separations, which is the subject of Sections 4 and 5 of this article, and second, by a combinatorial and statistical treatment of the subject which are so far incomplete and too lengthy to be described here. However, our initial assumption that such a

* Since preparing this paper the number of fields, groups and descriptors in ASTIA has been reduced through a revision.

** So far this second level of stratification was attained manually. Development of machine processes for the second level of stratification has just begun. The determination of the sub-groups is based upon associating descriptors, with inclusive descriptors in other attributes. The attributes in a single attribute-group may "overlap" in the sense that they have some values in common. Then inclusive attribute groups are forms consisting of one attribute from any attribute group.

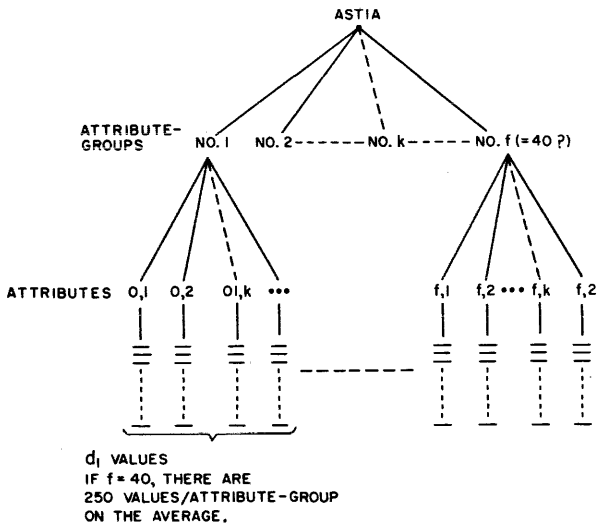


Figure 3. Schematic diagram of descriptor language with three strata that is proposed for possible use in ASTIA. The attributes are inclusive, but the values corresponding to a single attribute are exclusive.

separation is possible was based on statistics of descriptor distributions in descriptions of items and on the probabilities of existence of a solution to the descriptor separation problem.

Available ASTIA data indicates that the longest description of a document in that file consists of 20 descriptors.[1] Therefore, there exists 20 "inclusive" descriptors. These must then belong to at least 20 exclusive attributes. In Figure 3, 40 exclusive attribute-groups are assumed ($f = 40$) with an average number of values per attribute group corresponding to 250 (d_k average $= 250 = d_1 = d_2 = d_3$, etc.,) where f is a reasonably small number slightly larger than the minimum number of such attributes (which is for the case of ASTIA equal to 20). This arrangement is such that descriptor combinations which would contradict the exclusiveness of descriptors in each attribute group correspond to descriptions not actually used in the file. Since the latter consists of the great majority of such descriptions, intuitively the arrangement of descriptors in a relatively small number of exclusive attribute groups appears possible.

Human initiation of such a stratified descriptor language may or may not precede the machine process. The descriptor vocabulary may be stated in a table where the columns correspond to attributes. The human initiator of the file would examine a number of incoming items and assign the descriptors to specific columns, or the machine process, proposed in Section 4, can either initiate the assignment of descriptors to exclusive-attribute-groups or continue such a process after the initiation has been done by humans. This assignment is based entirely on the descriptions of the items coming into the file. Also, a human monitor can again use his judgment from time to time in transferring descriptors from one attribute group to another or to initiate new attribute groups. He can also create new descriptors by combining a number of descriptors previously used, into a new one, and by assigning this combination as a new value to a specific attribute.

A decrease in the total number of attributes f improves retrieval and item addition—deletion efficiency. This is indicated in the next section. Descriptors are assigned positions in a complete description (by belonging to a specific attribute group); the positional significance reduces the number of digits necessary for a description and it is sufficient to define such an individual descriptor only within an attribute group, instead of defining it in the larger vocabulary. In other words, the context within which a descriptor is stated conveys part of the information. Other advantages of such a system are evident from the application described in the next section.

3. FUNCTION OF DESCRIPTOR LANGUAGE STRATIFICATION IN INFORMATION RETRIEVAL

The block diagram of Fig. 4 outlines the tasks performed by the attribute assignment programs in the overall information retrieval process of the Multi-List system.[2, 3] The process is initiated by orders for retrieval (or storage). This is accompanied by an item description which may appear in natural language (Block 1 of Fig. 4). In Block 2 the input description is encoded in machine code in accordance with the Adjusted Vocabulary, which is an updated list of descriptors previously encountered. This encoding may utilize a tree structure which translates a natural language descriptor into an updated machine code.[4] As will be shown, the machine code may change frequently as it is designed to make machine operation efficient.

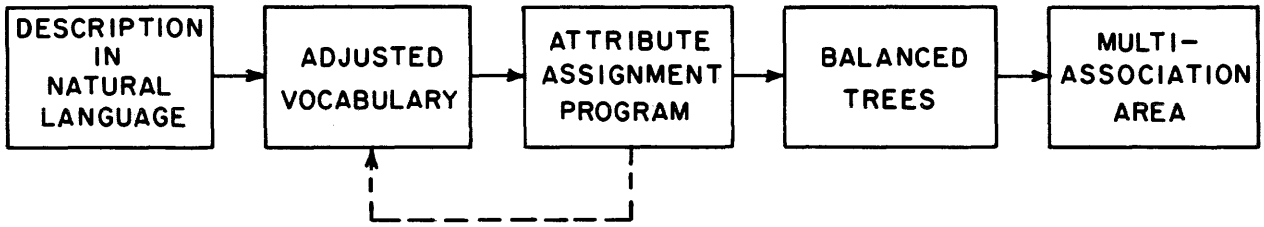


Figure 4. Gross flow chart showing the part of the Attribute Assignment Program in the Multi-List Information Retrieval process.

In the case of a new descriptor, which does not appear on the Adjusted Vocabulary, it is processed in Block 3 and then entered in the Adjusted Vocabulary in Block 2. A description input is referenced to the attribute assignment program block when two or more descriptors in the incoming description have previously been assigned to the same attribute group; i.e., previously they may have been considered "exclusive," while in the new item they are found to be "inclusive." Thus, there is a reassignment of such descriptors in Block 3, and then a correction of the Adjusted Vocabulary is called for.

Block 3 contains a program which takes as its input a new descriptor, or one that has been previously reassigned, and assigns it to an exclusive attribute group. It may happen that a previously assigned code will have to be reassigned. These "adjusted" codes, and additional codes which may change in the attribute assignment program, are fed back to the Adjusted Vocabulary. Attributes are assigned fixed positions in a description. A predetermined number of positions constitutes a partial description, which is used as a reference key. The item is then stored in a simulated associative memory as follows:[5] The keys are entered in respective tree structures in Block 4, together with the address of a corresponding item, heading a list of all items containing the given key in common. The tree performs a decoding function which locates the list for retrieval (or storage) in the *multi-association* area shown in Block 5.

The purpose of the Adjusted Vocabulary is to maintain freedom of changing machine code assignments of descriptors, without affecting the descriptor language used to communicate with the machine.

Consider a description to be an unordered set of descriptors. Each descriptor, in turn, is decoded by the Adjusted Vocabulary to a number pair i,j, where i is the attribute-group designator, and j is a serial index designating the value within the attribute group. If the given descriptor had appeared previously then it has a current i,j entry in the Adjusted Vocabulary, otherwise it is put aside until all descriptors in the description have either been assigned current i,j values or also have been put aside. If any two descriptors have been assigned the same i value in a current description, they are in conflict, since the same i value implies exclusivity while being in the same current description implies inclusivity. One of these conflicting descriptors is renamed to another column (attribute group) by a *Renaming Process* within the *Attribute Assignment Program* (Block 3). Its i,j code (and possibly others) changes and is immediately adjusted on the Adjusted Vocabulary (dotted line from Block 3 to 2). When all conflicts have been resolved the Attribute Assignment Program assigns the other new descriptors previously put aside to unused columns. It is to be noted that the renaming process, when required, must not introduce unresolved conflicts into the previous ensemble of descriptions in the system.

The Attribute Assignment Program assigns all input descriptors (initially) into f attribute groups. Note that the word "initially" is used in connection with f. This means that, it should always be possible to increase or decrease the number of attribute groups should it be found, either by machine or human, to be necessary.

4. THE AUTOMATIC ATTRIBUTE ASSIGNMENT PROGRAM*

The entire descriptor vocabulary may be arranged in a two dimensional matrix consisting of *columns* and *rows*. It is the object of this program to organize all input descriptions into f (initially) *columns*, where, each colum represents an exclusive attribute group.

The rule by which the exclusive attribute columns are formed is simply that all descriptors in a given column must be exclusive. The words "column" and "row" will become clear in the example which follows, where the exclusive descriptors are arrayed into columns, and inclusive descriptor groups are represented by rows.

The process for doing this is first briefly described and is then followed by a corresponding set of steps and by a flow chart.

For a given input descriptor there may be none or one or more exclusive columns to which it can be assigned. If there is one exclusive column then obviously the descriptor is placed there. If more than one exclusive column exists, then the descriptor may be assigned to any of them. This is called a *first order renaming* as it is performed in one step.

In the case where there are no exclusive columns to which the descriptor may be assigned there arises the necessity of shifting descriptors to other columns in order to provide an exclusive column. For instance, if a descriptor, (i,j) exists in a column, C_i, thus preventing C_i from being exclusive to the input conflicting descriptor, (i,j) may be moved to another column, C_k. By such a process (i,j) would become (k,j') and the input descriptor could be assigned to C_i thereby becoming (i,j). This does not mean that the input descriptor has the same attribute as the former descriptor in C_i, but that C_i now represents a different set of exclusive attributes. Such a process is called a *second order renaming* consisting of two steps, that of shifting descriptors from one column and that of first order renaming. It is also possible that the input descriptor be in column C_k, in which case it is swapped with (i,j); however, the renaming is still of second order. The strategy defining the order of renaming will be further discussed.

If for every C_i there is no descriptor (i,j) which can be shifted to some C_k, so as to make C_i exclusive to the input descriptor, then a third order renaming is attempted. Here a (k,j) in a column C_k can be shifted to an exclusive column C_m so as to make C_k exclusive to (i,j). Upon the shift of (i,j) from C_i to C_k, C_i becomes exclusive to the input descriptor, which can be assigned to C_i. This process is a third order renaming, and although it can inductively be extended to the *nth* order, the number of descriptors and descriptions involved soon become prohibitive in terms of processing time, for it must be realized that the (i,j) in the above scheme are actually sets, $(i,1)$, $(i,2)$ . . . (i,s), of descriptors which must be shifted, rather than individual descriptors, and each descriptor in this set may be renamed to a different column.

The Transition Table

The process and the order of a renaming can be systematically defined through the use of the *Transition Table* illustrated in Figure 5.

The rows in the table consist of all descriptions in the file involved in the conflict and renaming, up to but not including the conflicting description. Previously processed exclusive descriptors of the descriptions fall into specified columns of the table. A conflict is created when two descriptors in a given description (in item 6 of the example) have previously been assigned to the same column. This is resolved by a process which successively finds sets of descriptors in a given column, which if renamed to other columns would enable one of the conflicting descriptors to be renamed. These sets are called *transition sets*, T_p, and contain the descriptors of some column which enter into the *pth* transition. The maximum of p is the order of the renaming. Transition arrows are drawn in the *Transition Table* (Figure 5) for

* There are a number of ways to solve the renaming problem. The method selected here lends itself to a heuristic which directs a search for certain goals although it is not exhaustive. Other methods which systematically scan all combinations of a given attribute-value configuration have also been considered, but have been rejected on the basis of uncertain convergence properties. Future research is intended to contrast the efficiency of these latter techniques with the heuristic approach outlined in this paper. Prominent among the combinatorial techniques is the Walker backtrack method;[7] the S_k sets and a linearly ordered set A, both required by the Walker algorithm, have been defined by the authors for the stratification problem.

COLUMN / ITEM	1	2	3	4
1	1,1 —3→	2,1 —2→		4,1
2		2,1 ←1— 2 —2→	3,1	4,2
3	1,1 —3→	2,2 —2→	3,2 —1→	
4	1,1 —3→		3,2 —1→	
5		2,2 —2→		4,2
6	1,1 / 1,2			4,1

Figure 5. *Example*: Third Order Renaming
Attribute-Value Table for a Third Order Renaming
$T_1 = [(3, 1), (3, 2)]$
$T_2 = [(2, 1), (2, 2)]$
$T_3 = [(1, 1)]$

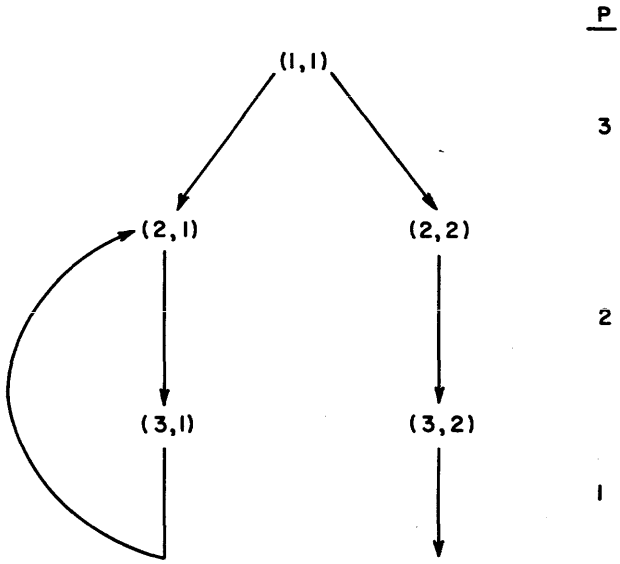


Figure 6. A "Tree" Representation of the Third Order Solution of the Example.

each set T_p, and correspondingly numbered with the value of p.

The example in Figure 5 illustrates a third order renaming, where the conflicting descriptors are (1,1) and (1,2) in item 6. (The example is being used for illustration and is not intended to represent the minimum order solution for this conflict.)

Alternatively the transition arrows can be represented as an n level "tree" of *goals* and *sub-goals*, which immediately makes evident an exponential increase in memory requirement with increasing renaming order. The tree representation shown in Figure 6 also suggests more clearly the structure of the program needed to effect an *nth* order renaming. However, when swapping is involved, as in the example, the "tree" can branch upward as well as downward.*

Starting from the top of the "tree," the primary goal is to rename the conflict descriptor, (1,1). The first two subgoals are the renaming of (2,1) and (2,2). A subgoal of (2,2) is the renaming of (3,2) which has no further subgoals since it can be renamed first order. But the renaming of (2,1) has the subgoal (3,1) which in turn has the subgoal (2,1), thus indicating a swap. The simplicity of the tree, at this point, belies the complexity of the program because (3,1) can only be swapped with (2,1) if it does not obstruct any higher goals, namely the (1,1)—(2,1) goal. If, for example, (3,1) had occurred in the description of the 1st item, this swap would not be possible.

Some upper limit on the order, n, should be set in a computer program, though n may be a parameter controlled by some performance index. Any conflict reaching that order would be considered an *absolute conflict*. At this point the number of exclusive attribute sets, f, may be increased by one, or the input of the description which is not resolved may be shelved for the time being or recourse is made to a human monitor. In time, the number of attribute groups, f, may thus increase.

At any given time there will be a number of errors in the system, in the sense that a human would not consider as logically exclusive all of the descriptors classified by the machine as exclusive. But the machine can only relate what it has seen and it is the entire body of preceding information (descriptions) which determines in general the state of the system. Use can also be made here of the human, who periodically monitors the system and introduces

* Insofar as trees only branch downward, the above representation is not a tree in the true mathematical sense. The terminology of goals and subgoals and tree description is borrowed, though not necessarily equivalent to, from reference 6.

changes to remove what appear to him to be errors. He effectively adds a "correlative impulse" to the system which reduces or eliminates the error. Note, however, that the actions of the automatic program are intended likewise to make the system tend toward an errorless state, but that its success depends upon the correlation of all past descriptive data and the chronological sequences of its inputs.

5. PROGRAM DESCRIPTION IN FLOW CHARTS AND CORRESPONDING STEPS

The above renaming program is further described by the flow charts for first and second order renaming in Figs. 8 and 9. The renaming program is part of a larger process, the Attribute Assignment Program, described in Fig. 7. In the following, the flow charts are explained with the aid of steps corresponding to the numbered boxes in the flow charts.

The input is a description D which consists of a sequence of descriptors $d_1, d_2, \ldots, d_k, \ldots, d_r$. Two constants appear in the program, k_1, and k_2. k_1 is the allowable number of renamings, and k_2 is a "0" or a "1" indicating recourse to a human monitor or a continuation of program control calling for an increase in the number of columns, respectively. f indicates the total number of columns at a given time, and the constants are subject to change by the machine operator during the course of the program.

The process steps of the Attribute Assignment Program as outlined in the flow chart of Fig. 7, are as follows:

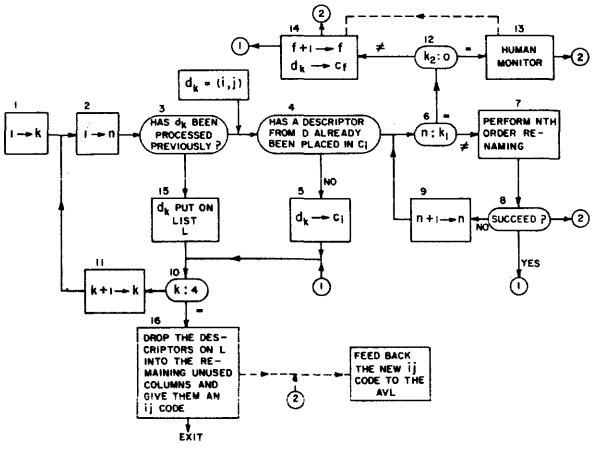


Figure 7. Flow Chart of the Attribute Assignment Program.

1. index the descriptors 1 through r. Set $k = 1$

2. n indexes the order of renaming. Set $n = 1$

3. Has descriptor d_k been processed previously? If the answer is in the affirmative, the Attribute Assignment Program had previously given the descriptor a code (i,j), indicating that it was placed in the ith column and was the jth descriptor in the column. In this case, go to step 4. If d_k has not been processed previously, go to step 15.

4. Since the descriptor name (i,j) indicates that its previous column was C_i we try to place it there; however, if some preceding descriptor in the current description $(d_1, d_2, \ldots d_{k-1})$ was placed in C_i, then two descriptors that were previously assumed exclusive are now found to be inclusive. Therefore, if a descriptor from D has not already been placed in C_i, go to step 5, otherwise, go to step 6.

5. Put d_k into C_i and go to step 10.

6. Have k_1 renamings been performed? If not, go to step 7. If so, go to step 12.

7. Perform an nth order renaming. The renaming is required because two (or more) of the descriptors in D are in the same column. Let these two descriptors be denoted (i,j_1) and (i,j_2). (If there are more than two descriptors in a column then the following applies to the entire set of such descriptors taken pairwise.) Figures 8 and 9 show first and second order renamings respectively. (The details of a general micro program flow chart for an nth order renaming are omitted due to space limitation.)

8. Has the renaming succeeded? If not, go to step 9, otherwise go to step 10, and also feed the new i,j code back to the Adjusted Vocabulary.

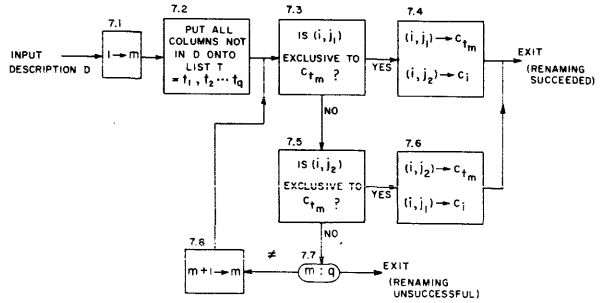


Figure 8. First Order Renaming.

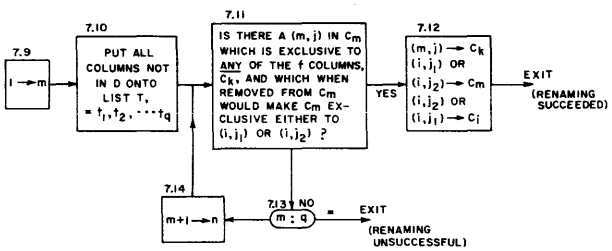


Figure 9. Second Order Renaming.

9. Increase n by 1 and return to step 6.

10. Have all r descriptions (d_1, d_2, . . ., d_{k-r}) been placed? If not, go to step 11. If so, go to step 16.

11. Increase k by 1 and go back to step 2.

12. Is k_1 set to "0," calling for a human monitor? If so, go to step 13 otherwise go to step 14.

13. A human attempts to make an appropriate renaming so as to make some column exclusive to d_1. If he cannot do so, then he returns control to step 14. If he succeeds, then the program feeds back all new i,j codes to the Adjusted Vocabulary.

14. The number of columns is increased by one and d_k is put into this new column. The new i,j code is fed back to the Adjusted Vocabulary and the program returns to step 10.

15. This descriptor has never been processed by the Attribute Assignment Program before and must be assigned a column and a position in the column, *i.e.*, an i,j code. These assignments are made, however, after all those descriptors that have an i,j code have been placed; therefore, this descriptor is put aside onto a list L, and the program returns to step 10.

16. All of the i,j coded descriptors have been assigned places, and what remains is to drop the descriptors put aside on the list, L, into the remaining unused columns. There have to be enough columns available because the maximum description length always determines the minimum number of columns.

6. A HAND SIMULATED EXAMPLE

In the following example a list of seven items are to be encoded into exclusive attribute groups. There are four tables used to aid in the explanation of the process. Table 1 presents the seven input items as they would appear at the machine input, in natural language; however, for conciseness, two digit decimal numbers are used to represent natural language descriptors. Table 2 consists of the Adjusted Vocabulary. The numbers without parentheses indicate the i,j codes of the descriptors at the end of the processing of the seven items. The parenthesized codes are the intermediate codes assigned during the process and later renamed. The i,j code consists of four digits; the first two designate i and the last two j.

TABLE 1

Descriptions Corresponding to First Seven Input Items in Chronological Order.

1. 11, 19, 32
2. 32, 25, 08, 18
3. 11, 18, 25
4. 41, 16, 08
5. 11, 41, 34, 25
6. 16, 34, 25, 08
7. 16, 32

TABLE 2

Adjusted Vocabulary

Natural Language Descriptor	i,j codes	
11	0101	
19	0201	
32	0301	
25	(0102)	0501
08	(0202)	0102
18	0401	
41	(0102)	0402
16	(0302)	0202
34	(0203)	0302

Table 3 shows the placement of descriptors into appropriate columns. The i,j codes are used to establish the placement in Table 3, but only the j part of the i,j code is retained since the column indicates "i." However, in the Adjusted Vocabulary the entire i,j code is recorded. As an aid in following the problem, the natural descriptor name is included in parentheses in Table 3.

There is a short discussion on the entry of each item in Table 3, that relates the algorithm to the flow chart in the preceding section (Figure 7). Table 4 is auxiliary to the discussion and shows the input items with their current i,j codes (current at the time of addition of each item). Also Table 4 is effectively a synthesis of the information in Tables 1 and 2, and it is inserted for convenience of explanation.

TABLE 3

Attribute Assignment Columns

Column Items	1	2	3	4	5	After Addition of Item
1	01	01	01			1
1	10(11)	01(19)	01(32)			2
2	02(25)	02(08)	01(32)	01(18)		
1	01(11)	10(19)	01(32)			
2		02(08)	01(32)	01(18)	01(25)	3
3	01(11)			01(18)	01(25)	
1	01(11)	01(19)	01(32)			
2		02(08)	01(32)	01(18)	01(25)	
3	01(11)			01(18)	01(25)	4
4	02(41)	02(08)	02(16)			
1	01(11)	01(19)	01(32)			
2		02(08)	01(32)	01(18)	01(25)	
3	01(11)			01(18)	01(25)	5
4		02(08)	02(16)	02(41)		
5	01(11)	03(34)		02(41)	01(25)	
1	01(11)	01(19)	01(32)			
2	02(08)		01(32)	01(18)	01(25)	
3	01(11)			01(18)	01(25)	
4	02(08)		02(16)	02(41)		6
5	01(11)	03(34)		02(41)	01(25)	
6	02(08)	03(34)	02(16)		01(25)	
1	01(11)	01(19)	01(32)			
2	02(08)		01(32)	01(18)	01(25)	
3	01(11)			01(18)	01(25)	
4	02(08)	02(16)		02(41)		7
5	01(11)		02(34)	02(41)	01(25)	
6	02(08)	02(16)	02(34)		01(25)	
7		02(16)	01(32)			

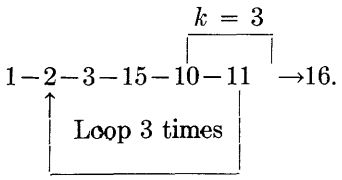
TABLE 4

Auxiliary Table to Discussion of Table 3

Item Added	Descriptors Natural Lang.	*ij*	Natural Lang.	*ij*	Natural Lang.	*ij*	Natural Lang.	*ij*
1	11	New	19	New	32	New		
2	32	0301	25	New	08	New	18	New
3	11	0101	18	0401	25	0102		
4	41	New	16	New	08	0202		
5	11	0101	41	0102	34	New	25	0501
6	16	0302	34	0203	25	0501	08	0202
7	16	0302	32	0301				

Discussion of Table 3 (Entry by Entry)

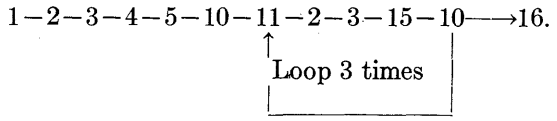
1. The first description contains three new descriptors, none having been processed previously; therefore, $r = 3$ in the flow chart of Figure 7. All of the descriptors would go onto list L (Block 15) and would then be distributed among the first three (unused) columns. The sequence of blocks (Figure 7) traversed in this assignment is as follows:



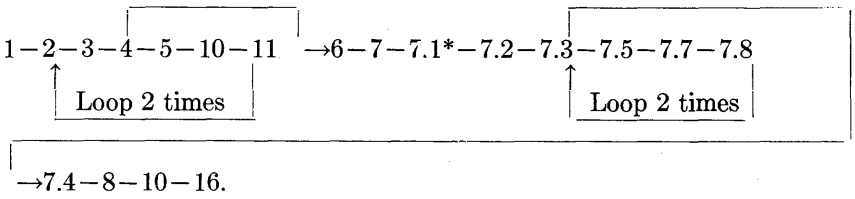
The number sequence is read from left to right while loops are indicated by reverse directed arrows. The number of times a loop is traversed (including the first pass) is marked. The loop exit is indicated by a line out of the top or bottom of some step within the loop.

2. In the second item, descriptor 32 has an assigned name of 0301 and hence can be assigned immediately to column 03. The remaining 3 descriptors are new and are dropped arbitrarily into columns 01, 02, and 04. The descriptors receive an assigned code when they are dropped into columns, and these codes are fed back to the Adjusted Vocabulary.

The sequence of process flow chart steps used to assign item 2 are:

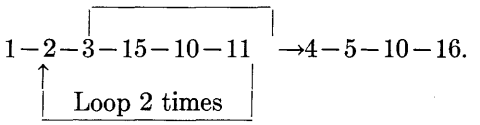


3. In item 3 all three of the descriptors have previously been assigned codes, but descriptors 11 and 25, which were previously exclusive and were in column 01 are now inclusive because they both appear in the same description. A first order renaming is sufficient to separate the two inclusive descriptors. Since column 04 is a part of the item (descriptor 18) either 11 or 25 can be moved to columns 02, 03, or 05. Only column 05 yields to a first order renaming; hence, descriptor 25 (0102) is renamed to column 05 and becomes 0501, and the Adjusted Vocabulary is adjusted by erasing the code 0102 for descriptor 25 (denoted by parenthesized 0102 in Table 3) and entering 0501. Descriptor 11 (0101) then remains in column 1 and descriptor 18 (0401) in column 04. The process flow chart steps are:

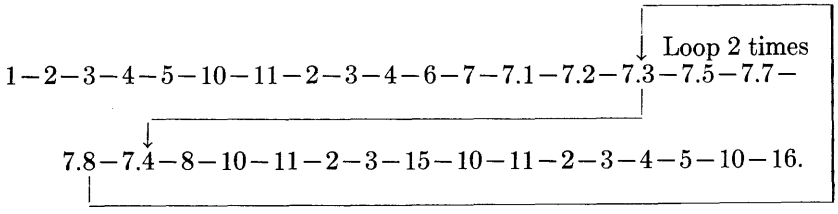


4. In item 4 descriptor 08 can be dropped into column 02 again without conflict, and descriptors 41 and 16, which have never been assigned a code are dropped into columns 01 and 03. The reader must note that Table 3 is only a schematic used as a guide in explaining the process. The information contained in that schematic is stored explicitly in the multi-association area as the assigned descriptor code number. Hence, when we say that a descriptor is "dropped" into a column without conflict it merely means that an encoding on a tree can be initiated immediately without a renaming.

The process steps for the entry of item 4 are:

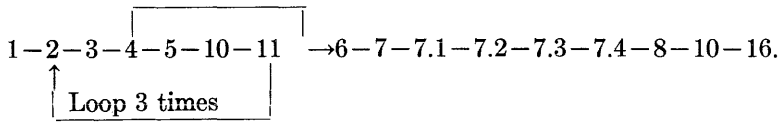


5. In item 5 descriptors 11 and 41 are seen to be inclusive. A first order renaming again suffices to remove the conflict. Descriptor 41 (0102) is renamed to column 04 (becoming 0402)). The process steps are: (Let $j_1 = 02$, $j_2 = 01$, and $t_1 = 02$, $t_2 = 03$, $t_3 = 04$.)

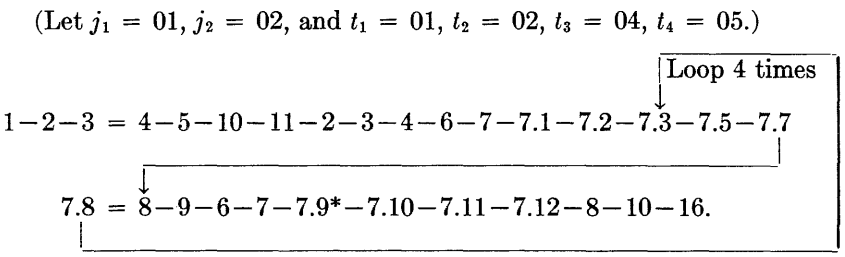


* Here block numbers refer to Fig. 8 for details of the renaming process. Let $j_1 = 02$, $j_2 = 01$, $t_1 = 02$, $t_2 = 03$, $t_3 = 05$.

6. The process steps for entering item 6 are:



7. Here for the first time we encounter a second order renaming. Descriptors 16 and 32 must be separated, but both are inclusive to every other column. The second order renaming is effected by shifting descriptor 34 (0203) from column 02 to 03 (renaming it to 0302) and 16 (0302) from column 03 to 02 (renaming it to 0202). Then descriptor 32 (0301) can be dropped into column 03 without conflict. The process flow chart steps are:



7. CONCLUSIONS

The strategy described here has been programmed for the IBM 7090. Experiments, to be described in a separate report used first artificially prepared input descriptions and a limited amount of ASTIA live data. In all these cases the machine was able to discover rapidly a predetermined or satisfactory stratification. More extensive experiments using ASTIA data are currently in process.

REFERENCES

1. *Optimization and Standardization of Information Retrieval Languages and Systems.* Technical Status Report No. 2, Contr. AF49 (638)835, Applied Mathematics Department, Remington Rand Univac, Blue Bell, Pa., June 1961. Also see Status Report No. 1, Jan. 1961 for procedure used to schedule technical conferences.
2. *The Multi-List System Technical Report No. 1.* Contr. NONR551(40), The Moore School of Electrical Engineering, November 1961.
3. PRYWES, N. S. and GRAY, H. J., *The Multi-List System for Real Time Storage and Retrieval,* Proc. of the IFIP Congress 1962, VIII, 1, p. 112-116, August 1962.
4. LANDAUER, W. I. and PRYWES, N. S., *A Growing Tree for Descriptor Language Translations,* Proc. of the 1962 Symposium on Symbolic Languages in Data Processing, March 1962.
5. PRYWES, N. S. and GRAY, H. J., *The Organization of a Multi-List Type Associative Memory,* Gigacycle Computing Systems. AIEE General Meeting, Jan. 1962, p. 87-101.
6. NEWELL, A., SHAW, J. C., and SIMON H. A., *Report on a General Problem—Solving Program,* The RAND Corporation Paper, P-1584, January 1959.
7. WALKER, R. J., *An Enumerative Technique for a Class of Combinatorial Problems,* Proc. of Symposia in Applied Mathematics, Vol. 10, p. 91, 1960.

* Here reference is made to the blocks in Figure 9. $m = 2, j = 3, k = 3$
$(i, j_2) = 0302$ is exclusive to C_{02} when 0203 is removed.

A COMPUTER APPROACH TO CONTENT ANALYSIS: STUDIES USING THE GENERAL INQUIRER SYSTEM

Philip J. Stone, Harvard University
Earl B. Hunt, University of Sydney

The General Inquirer[1] is an IBM 7090 program system that was developed at Harvard in the spring of 1961 for content analysis research problems in the behavioral sciences. The first part of this paper describes this system and how it has been used. During the summer of 1962, the General Inquirer was merged with the Hunt Concept Learner[2, 3] to produce a method for automatic theme analysis[4]. The second part of this paper discusses the rationale behind this development and some recent signs of its future promise.

Within the behavioral sciences, much of the raw data to be analyzed consists of written text. A psychologist, for example, may hand you an inkblot and ask you to describe what you see. A public opinion interviewer may ask for your free answer to his questions. A sociologist may record conference group processes and make transcripts of tape recorded sessions. A political scientist may collect diplomatic notes. In each case, the data is the same: written material such as you are reading right now. From the viewpoint of the behavioral scientist, this is raw data, consisting of words and punctuation marks recorded on a page. The analysis is yet to be done.

As defined by Berelson[5] in 1952, "Content analysis is a research technique for the objective, systematic and quantitative description of the manifest content of communication". Berelson uses the term "objective" to indicate that the procedure should be explicit, one that can be replicated exactly by other analysts. "Systematic" means that "*all* the relevant content is to be analyzed in terms of *all* the relevant categories" in order to secure unbiased information for the hypotheses being tested. "Quantitative," of course, refers to the process of counting positive or negative instances. Finally, in order to do all this, the procedure has to be based on the "manifest" aspects of the text; however, as Berelson points out, "the results of content analysis frequently serve as a basis for the 'interpretation' of latent content".

The General Inquirer was developed to help further the rigor of these procedures. In describing a content analysis procedure to a computer, nothing may be left implicit for processes of intuition. The program is an "objective" description of the content analysis process, it must operate on "manifest" features of the text, and it can be designed to yield carefully counted "quantitative" results. Once the computer handles the task, the program is "systematic" in carrying out all of the details of the analysis. Since a systematic program may analyze many variables at once, one may discover trends that are indeed "latent" to casual observation.

Verbal text may be regarded as the result of many different psychological processes acting at once, each revealing its influence in the final product in different subtle ways. Our classification procedures in the General Inquirer are concerned with recurrently expressed or assumed values, underlying types and intensities of motivation, perceived demands of the environment, and institutionalized structuring of both

"This work was supported by NIMH (JSPH M-4169) and ONR (TOSK 047-003). IBM 7090 computing has been done with the cooperation of the computation centers at MIT, Harvard, U.C.L.A. and Stanford."

demands and action as these may be shown directly or indirectly by text materials.

One common purpose of the General Inquirer is to find psychological indexes that will discriminate between two text sources. While making such discriminations is indeed a problem in identifying authorship, it differs from the classical authorship problems (Federalist papers, Shakespeare-Marlowe, books of Homer, etc.) in several respects. First, each source is usually not a single author, but rather a group of people. Second, our goal is not just to make an efficient discrimination, but is also to gain further understanding of the psychological forces and perceived demands of the situation that were in effect when the document was written. If our only goal was to identify authorship, usually we could find our best clues in idiographic stylistic differences (noun declension preferences, "while" versus "whilst,"[6] etc.).

PROCEDURES

Text preparation. The alphanumeric characters of the text (including punctuation) are keypunched on IBM cards, each card roughly corresponding to a typewritten line with 80 space fixed margins. Between-card breaks come either between words or after a hyphen as in regular typewriting.

System operation. The text is transferred from the punched IBM cards to magnetic tape by an IBM 1401 machine. The 7090 computer then reads the alphanumeric characters and separates them into words and sentences. Certain regular word endings are removed and each word is looked up in a dictionary. If a word is found in the dictionary, tags indicating the word's membership in one or more categories specified by the investigator are attached to the sentence. If a word is not found in the dictionary, it is put on a leftover list for further examination by the investigator. The sentences, together with their tags, are then stored on binary tape for repeated use in inquiry procedures. The investigator uses inquiry procedures to ask for the number of times (and possibly the retrieval of each instance) where a particular combination of tags or specific text words co-occurred in the same sentence. Up to a hundred such questions may be processed on one pass of the binary tape. By arranging the questions into "question sets", it is possible to ask about various disjunctive relationships.

If the keypunched text is marked with a simplified form of syntactic coding, the investigator can make inquiries not only about the co-occurrence of certain text words and/or tags, but also can specify the syntactic relationships that must appear between them. The computer can also use the syntactic codes to help tag correctly many otherwise ambiguous words.

A General Inquirer dictionary may be considered as representing an operational explication of the scientist's theory or frame of reference. The investigator's questions represent the "rules" he develops within that theory for discriminating one kind of text from another. Contrary to hand content analysis procedures, both dictionary and question inquiries can be revised repeatedly without the necessity of recoding or repunching the original data.

Selecting tag categories. Since dictionary development is a crucial step in General Inquirer procedures, let us examine two different dictionaries currently in use. Figure 1 identifies the tag categories employed by our third general psycho-sociological dictionary[7] for analyzing texts of predominantly non-specialized vocabulary. Figure 2 gives the tag categories that Dr. Benjamin N. Colby*, an anthropologist, uses for studying themes in the folktales of different cultures.

Both Colby's dictionary and our own contain about 3500 entries each. Since the General Inquirer removes *s, es, ed, ly,* and *ing* suffix forms, the actual number of words that can be found in the dictionary is probably more than triple that number.

Our psycho-sociological dictionary makes a distinction between "first-order" and "second-order" tags: only one first-order tag may be used in categorizing a particular entry word, but one or several second-order tags may also be applied. All the tags in Colby's dictionary are first-order; each entry is associated with one tag and one tag only. Since multiple tagging of individual entry words can easily confuse later analyses, the first-order, second-order distinction should always be kept in mind.

* Dr. Colby is Associate Curator, Laboratory of Anthropology, Museum of New Mexico, Santa Fe, New Mexico. The dictionary and results referred to here are described in an article by Colby and Postal, currently in press in the journal, *Folklore*.

Figure 1. Third Psycho-Sociological Dictionary (Harvard)

FIRST ORDER TAGS

PERSONS
- self—all pronoun references to the personal self (I, me, mine, myself)
- selves—all pronoun references to the inclusive self (we, us, ours, etc.)
- other—all non-sex-specific pronouns for other (you, yours, they, theirs, etc.)
- male-role—all roles with specific male references
- female-role—all roles with specific female references
- neuter-role—all role names not connoting sex or occupations
- job-role—all roles with clear occupational reference, theoretically open to both sexes

GROUPS
- small-group—groups usually able to have face to face interaction
- large-group—collectivities usually too large for face to face interaction

PHYSICAL OBJECTS
- bodypart—parts of the body
- food—articles or types of food
- clothing—articles or types of clothing
- tool—instrumental objects or artifacts of any kind (broader category than hand tools)
- natural-object—objects not made by man (plants, animals, and minerals)
- non-specif-obj—abstract references to objects (connoting intellectualization)

PHYSICAL QUALIFIERS
- sensory-ref—smells, colors, tastes, etc.
- time-ref—references to measurement of time
- space-ref—references to spatial dimensions
- quantity-ref—references to units and measures of quantity

ENVIRONMENTS
- social-place—buildings and building parts; political, social, and economic locations
- natural-world—geographical places, weather references and cosmic objects

CULTURE
- ideal-value—culturally defined virtues, goals, valued conditions and activities
- deviation—culturally devalued goals, conditions, and types of activities
- action-norm—normative patterns of social behavior
- message-form—names of communication media in a very broad sense, including art objects and money
- thought-form—units and styles of reasoning

EMOTIONS
- arousal—states of emotional excitement
- urge—drive states
- affection—indicants of close, positive, interpersonal relationships
- pleasure—states of gratification
- distress—states of despair, fear, guilt, shame, grief, failure, or indecision
- anger—forms of aggressive expression

THOUGHT
- sense—perception and awareness
- think—cognitive processes
- if—conditional words
- equal—words denoting similarity
- not—words denoting negation
- cause—words denoting a cause-effect relation
- defense-mechanism—standard psychological terms for defense mechanisms

EVALUATION
- good—synonyms for good
- bad—synonyms for bad
- ought—words indicating a moral imperative

SOCIAL-EMOTIONAL ACTIONS
- communicate—processes of transmission of meaning
- approach—movement toward
- guide—assistance and positive direction
- control—limiting action
- attack—destructive, hostile action
- avoid—movement away from
- follow—submissive action

IMPERSONAL ACTIONS
- attempt—goal-directed activity, implying effort
- work—task activity
- get—obtaining, achieving action
- possess—owning, consuming
- expel—ejecting

SECOND ORDER TAGS

INSTITUTIONAL CONTEXTS—specification of the social context of roles and actions

- academic
- artistic
- community
- economic
- family
- legal
- medical
- military
- political
- recreational
- religious
- technological

STATUS CONNOTATIONS—male-, female-, neuter-, and job-role status implications

- higher-status
- peer-status
- lower-status

PSYCHOLOGICAL THEMES

- overstate—emphatic or exaggerative words, generally adjectives or adverbs (connotes a defensive style)
- understate—words, generally adjectives or adverbs, connoting doubt or uncertainty (connotes a defensive style)
- sign-strong—words connoting strength or capacity for action
- sign-weak—words connoting weakness or incapacity for action
- sign-accept—words implying interpersonal acceptance
- sign-reject—words implying interpersonal rejection
- male-theme—psychoanalytic symbols of masculinity
- female-theme—psychoanalytic symbols of femininity
- sex-theme—direct or indirect references to the sex act
- ascend-theme—words associated with rising, falling, fire, and water, indicating concerns related to the Icarus complex
- authority-theme—words connoting the existence or exercise of authority
- danger-theme—words connoting alarm or concern with danger
- death-theme—words connoting dying, end

Figure 2. Anthropological Dictionary (Colby)

1. Derived From Kluckhohn Value Categories
   - Determine, Order, Indeterminate
   - Dominate, Follow, Leader, Power, Agree
   - Fame, Pride, Equality, Status
   - Good, Evil, Suspicion
   - Assist, Empathy, Guest, Selfish, Rivalry, Trick, Scapegoat
   - Alone, Gregarious, Withdraw
   - Choice, Individual, Group, Reject, Punish
   - Chance, Curiosity, New, Protect, Caution
   - Independence, Dependence, Ask
   - Self-control, Over-indulge, Abstain
   - Rational, Wise, Truth, Unknown, Naive, Foolish, Anger, Happy, Sad, Enthusiasm, Amuse, Dislike, Fear
   - Tense, Relaxed
   - Unique, General
   - Quality, Quantity

2. Perception and Communication
   Aware, See, Smell, Talk, Taste, Sound, Quiet, Bright, Dark, Heavy, Hard, Hot, Cold, Color, Texture, Form
3. Space and Time
   Little, Big, High, Low, Narrow, Wide, Fast, Slow, Place, Time, Now, Old, Permanent, Future, Past, Building, Road
4. Self Identity
   Bodypart, Beauty, Bathe, Clothing, Healthy, Sick, Pain, Ugly, Ornament
5. Nature
   Sky, Earth, Air, Fire, Fluid, Dirt, Weather
6. Sex and Kinship
   Genital (Male and Female), Sex, Male, Malesymbol, Female, Marriage, Birth, Kinship, Kinship Affinal
7. Activities
   Hunt, Husbandry, Fish, Farm, Manufacture, Magic, Ritual, Exchange, Build, Repair
8. Miscellaneous Motif Groupings
   Able, Accomplish, Goal, Get, Keep, Increase, Want, Work, Plan
   Arrive, Leave, Move, Go Kinesthetic, Change
   Rest, Sleep, Difficult, Easy, Lazy, Unable, Fail
   Cover, Container, Complete, Empty, Full, Hole, Imperfect, Include, Uncover, Reveal, Release
   Boundary, Break, Cut, Pierce, Tear, Tie
   Anal, Oral, Food
   Death, Danger, Ruin, Lost, Secret, Prevent
   Money, Ownership

Our psycho-sociological dictionary uses 83 tags. With the exception of the categories "self", "selves", and "other", no tag category contains less than twenty words. Some second-order tags are used in categorizing over 300 words. While the number of tag categories is relatively small, further categorizing specificity is gained by considering those subgroups formed by common membership under both a first-order and a second-order tag. For example, those words matching both "job-role" and "legal" is a list in itself, quite different from those words matching both "job-role" and "academic" or "legal" and "ideal-value". These groups of words defined by first-order, second-order intersections can be directly specified in question retrieval procedures.

Inasmuch as sociological and psychological concerns can be separated, those tags with primary reference to the socio-cultural realm are listed in the left hand column of figure 1, those emphasizing psychological processes and themes are given in the right hand column. The sociological and anthropological concepts of roles, collectivities, actors, situations, values, norms, institutions, etc., are represented by tags under the headings entitled persons, groups, physical objects, qualifiers, environments, culture, institutions, and statuses. The psychological concepts of emotion and cognition, interpersonal and instrumental behavior are represented by the tags listed under the headings emotions, thought, evaluation, social-emotional actions, impersonal actions, and psychological themes.

If the distinction between the two columns is then combined with the distinction between first- and second-order tags, four distinct areas emerge. The first-order tags in the left hand column consist mainly of objects defined in sociological terms. The second-order tags in the left hand column refer to the social structure of society, specifically to institutional and status divisions. By contrast, the first-order tags on the right indicate basic psychological processes. The second-order tags on the right refer to some of the underlying psychological motivations of personality. Within each quadrant, the tags are arranged in sets. Wherever

possible, the tags within a set are arranged in an order approximating a progression from the more personal and intimate to the more impersonal and objective.

The Colby anthropological dictionary uses 180 tags. The number of entry words for each tag ranges from 4 to 42. Many of the tag categories were developed as part of an attempt to validate cross-cultural ratings using the Clyde Kluckhohn value categories.[8] The tag categories have been built up through a substruction of the Kluckhohn binary categories (such as Good-Evil, Determinate-Indeterminate) into more precise units. In addition to the Kluckhohn categories, other motifs and themes which appeared to be important in the preliminary content analysis of folklore texts have been added. The tag categories are arranged under very general headings in figure 2 for clarity of presentation.

It should not be thought that all General Inquirer dictionaries must contain a large semi-comprehensive list of tag categories. For example, a General Inquirer dictionary[9] recently completed by the Stanford University project on International Conflict and Integration uses only tag categories relevant to the three major dimensions of Osgood's Semantic Differential[10]; namely, good-bad, strong-weak, and active-passive. Each of these dimensions has tag categories representing six levels of intensity, thus making a total of 18 different tags in all.

EXAMPLE APPLICATIONS

1. Comparing overt versus latent trends in folklore material.

The data. Approximately 12,000 words of folklore text from each of ten cultures were prepared for General Inquirer analysis. The ten cultures are: Kwakiutl, Egypt, Eskimo, India, China, Baiga, Russia, Kikuyu, Thailand, and Japan. Usually, such folktale materials have been gathered and recorded by missionaries and anthropologists. English translations, of course, have been used for purposes of the analysis.

The problem. Colby is interested in discovering themes which both characterize and distinguish cultures. More ambitiously, it is hoped that theme clusters may be discovered which provide insight into the way cultures or subcultures cognitively structure the world.

One problem of particular interest is the relationship between latent and overt themata in folklore material. In the results reported here, Colby compared the frequency of words denoting overt sexual activity with the frequency of words connoting words of latent sexual reference. Similarly, comparisons were made between overt and latent orality. Colby's latent sex index included references to possible symbols of the male sexual organ, to piercing or thrusting actions, and to possible symbols of the female sexual organ. The overt sex index included direct references to affection and sexual acts. The overt oral index consisted of words referring directly to the oral body zone and oral processes. The latent orality category is made up of what are thought to be symbolic references to ingestive states (full, empty, etc.); it is admittedly unclear, however, exactly how this latent oral category would relate to a latent anal syndrome.

The results. The rank ordering of the ten cultures on these four measures is shown in figure 3. As can be seen by inspection, the rank order for overt sex is inversely related to the rank order for latent sex ($r = .55$, prob. $< .06$), while the rank order for overt orality is posi-

Figure 3. Rank Order of Folktales on Four *Tag* Indices

	Low Occurrence	*High Occurrence*
Overt sex:	Eskimo, China, Kwakiutl, Kikuyu, India, Russia, Japan, Baiga, Egypt, Thailand	
Latent sex:	Thailand, Baiga, Egypt, India, Kikuyu, Kwakiutl, Eskimo, China, Japan, Russia	
Overt orality:	Kwakiutl, Japan, India, Egypt, Thailand, China, Kikuyu, Russia, Eskimo, Baiga	
Latent orality:	Kwakiutl, Egypt, Thailand, Eskimo, Japan, India, Russia, China, Kikuyu, Baiga	

tively related to the rank order for latent orality ($r = .66$, prob. $< .05$). With regard to sex, the implication seems to be that if open sexual references are inhibited in a culture, the sexual motives will nevertheless find indirect outlets. With regard to orality, several implications are suggested. Perhaps our latent category is not really latent. On the other hand, perhaps orality is not subject to processes of inhibition. Cultures low on orality are perhaps simply not making an issue of it, possibly because of their better nursing and weaning practices. Within a culture, sex is apparently always present in one form or another, whereas infant orality may be at least somewhat resolved.

2. Distinguishing real from simulated suicide notes.

The data. With the cooperation of the Coroner of Los Angeles County, Dr. Shneidman of the Los Angeles Suicide Prevention Center has collected 721 suicide notes from the court records of all recorded suicides for the ten year period 1945-1954. In each year, between 12 and 15 per cent of those committing suicide left notes. These notes come from both sexes (almost three males to every female), with the individuals ranging from twenty-five to fifty-nine years in age.

Simulated suicide notes were obtained from persons contacted in labor unions, fraternal groups, and the general community. Subjects were instructed as follows:

> "A study is being done on the prevention of suicide. For this, it is necessary to obtain many suicide notes written by normal people. For this reason, you are asked to write below, in your own words, the suicide note that you would write if you were going to take your own life. Make your note sound as real as you possibly can. Write what you think *you* would write if you were planning to commit suicide. Before you write the note, answer these two questions first:
>
> a) What method would you use to take your own life?
>
> b) To whom would you address the note you are writing?"

In order to keep the two groups homogeneous (and to emphasize whatever differences might exist in the notes), all sixty-six real and simulated notes were selected from those written by individuals who were male, Caucasian, Protestant, and native born. Each of the thirty-three simulated note writers was matched, man for man, with a real note writer who was not only of similar age (within five years), but also of the same occupational level.

The problem. Our research was twofold. As an academic exercise, we wanted to test whether a set of General Inquirer measures could be developed that together would effectively discriminate between real and simulated notes. Second, we were interested in what meaningful insights the Inquirer could offer as to the differences between these two groups.

The correct identification of which note in a pair is real and which is simulated is not a trivial challenge. Osgood[11] reports giving this same task to eight graduate students in psychology and finding that they could do no better than chance . One of us (Stone) gave this task to six members of a sophomore tutorial at Harvard, the students having no reading background particularly related to suicide. As a whole, the Harvard sophomores did better than chance, the mean being 66 per cent correct, the best performance being 75 per cent correct. Let us consider the 66 per cent as a base reference. Could the General Inquirer do better?

The results. The original analysis was done in February, 1962, using a predecessor of the Harvard Psycho-sociological Dictionary described above.[12] The procedure for building and testing a discriminate function was as follows: the actual source (i.e. real or simulated) of each of the first fifteen pairs of notes was revealed to us by Dr. Shneidman. These notes were then compared using the General Inquirer. Three factors were found to discriminate:

1) References to concrete things, persons, and places (higher for real notes).
2) Use of the actual word "love" in the text (higher for real notes).
3) Total number of references to processes of thought and decision (higher for simulated notes).

A very simple discriminate function was then developed: the score on the third measure was substracted from the sum of the scores of the first two measures. This index correctly discriminated thirteen of the fifteen pairs of notes.

This discriminate function was then applied to the remaining eighteen pairs of notes, with the members of the research team not knowing which of these were real and which were simulated. After the predictions were made, Dr. Shneidman was again consulted. Seventeen of the eighteen pairs of notes had been identified correctly. This figure is quite significant when compared with chance expectation, the performance of human judges, and most attempts of other investigators to analyze this same data (cf. review of literature in article by Ogilvie et al.).[13]

While constructing a successful discriminate function was a complex task, a complete General Inquirer analysis of the differences between the two kinds of notes was of a much larger order. After our Third Psycho-sociological Dictionary was finished, the suicide notes were again processed through the computer. At this point, of course, our research staff was no longer naive, so we could no longer make independent predictions on half of our data. Figures 4a and 4b show the differences in the actual number of times certain tags appeared in a specific syntax position in each of the two kinds of text.

Figure 4a. Raw Tag Counts Higher for Real Notes

FIRST ORDER WORDS

Tag Label	*Syntax Position*	*Number of Times Applied to Text*	
		Real	*Simulated*
other	attribute	17	6
male-role	subject	45	12
male-role	object	17	6
male-role	attribute	5	0
female-role	subject	200	67
female-role	object	137	35
female-role	attribute	21	5
tool	subject	5	0
tool	object	10	2
non-specif-obj	object	47	19
sensory-ref	object	6	0
quantity-ref	subject	27	10
social-place	object	27	4
affection	verb	51	18
bad	subject	9	2
bad	verb	9	0
communicate	attribute	14	3
attack	verb	17	2
attempt	leftover	23	9
get	verb	46	16
possess	verb	20	6

SECOND ORDER WORDS

Tag Label	*Syntax Position*	*Number of Times Applied to Text*	
		Real	*Simulated*
family	object	38	15
religious	subject	10	2
higher-status	subject	13	4
higher-status	object	11	1
higher-status	leftover	14	4
male-theme	(all)	18	1
sex-theme	subject	17	4
sex-theme	verb	46	13
sex-theme	object	8	2

Figure 4b. Raw Tag Counts Higher for Simulated Notes

FIRST ORDER WORDS

Tag Label	*Syntax Position*	*Number of Times Applied to Text*	
		Real	*Simulated*
selves	(all)	17	23
time-ref	object	3	10
natural-world	subject	4	11
natural-world	object	5	17
ideal-value	object	2	8
thought-form	subject	1	10
thought-form	object	3	10
distress	object	5	12
think	object	3	12

SECOND ORDER WORDS

Tag Label	*Syntax Position*	*Number of Times Applied to Text*	
		Real	*Simulated*
academic	object	5	12
sign-weak	subject	19	27
death-theme	(all)	54	62

In order to understand the criteria for selecting the tag labels which appear in figures 4a and 4b, we must first consider differences in the lengths of the notes. While shorter notes might be either real or simulated, most longer notes were real. This causes the thirty-three real notes to have a considerably greater combined length (total number of text words = 4112) than the simulated notes (total number of text words = 2542). Thus, unless the differences in raw frequency count for a particular tag label are considerably more or less than the differences in overall length, we are not impressed. Our criteria reflect this:

1) Select for figure 4a all cases where the total frequency of real suicide note tags is at least 2.5 times greater than the corresponding simulated note frequency, providing there is a minimum difference of five.
2) Select for figure 4b all cases where the total frequency of the simulated suicide notes is at least five counts more than the corresponding count for the real suicide notes.

Five is an arbitrary selection, reflecting what we feel to be a minimum difference to be of interest. Any difference in favor of the simulated notes is against odds (since the simulated notes are shorter), thus a simple difference of five is all that is required.

Figures 4a and 4b show a number of tag count differences which might be used in building discriminate functions. No information, however, is given on the distribution across notes. In actuality, some tag count differences are spread across many notes while others are concentrated in just a few. A tag may be highly discriminating for a few note pairs and be irrelevant in discriminating many other pairs. To account for this, we reason that there are multiple factors involved in suicide notes, some being a significant cue in one pair, others being a significant cue in another. If a cue is irrelevant to a particular pair of notes, then its prediction should be randomly determined. If, however, the cue is relevant, it should tend to predict in the correct direction. Adding together the different cues is like adding together signals in a background of noise. In some cases, one cue will be the signal while several others are noise; in another case, several of the other cues may serve as signals while the first cue functions as noise. In a similar vein, we can afford to have a limited amount of "mistagging" in our procedures, providing it acts like random noise in our analyses. If enough cues are added together, the signals usually emerge from the noise and a correct prediction can be made.

The syntax positions shown in figures 4a and 4b are "subject", "verb", "object", "attribute", "leftover", and "all" (for all syntax positions combined). These syntax marking procedures are described in detail elsewhere[1]. The terms "subject", "verb", and "object" are used in a sense similar to, but slightly broader than, the meanings you learned in grammar school.[11] "Attribute" indicates those words in a phrase indicating the source of a statement, such as, "He says . . .", "She knows that . . .", "It is true that . . .", etc. "Leftover" refers to counts for syntax positions not listed here and for words lacking syntax marks. As we shall see, reporting tag counts for each major syntax position separately is a useful aid in analyzing the data. Since the counts for each syntax position are based on only a few words per sentence, a separate analysis of each syntax position helps keep counts from being determined by multiple occurrences in just a few sentences.

The information in figures 4a and 4b can be combined with retrieval procedures to gain more information about the text. For example, the tag "possess" is shown on the bottom of the right hand column of figure 4a as being used in the verb position more frequently in real notes than in simulated notes. Who is seen as doing the possessing? We see further up in this same column that there are several likely possibilities. Male-role and female-role, for example, are both relatively high in the subject position for real suicide notes. If we make the retrievals "male-role/subject—possess/verb" and "female-role/subject—possess/verb", we find that it is the sentences containing female rather than male references as subject that are causing high counts. The retrieval for the question "male-role/subject—possess/verb" consists of only one sentence and this from the text of a simulated note:

note 15, sentence #11
GOD BLESS AND KEEP YOU BOTH.

The retrieval for "female-role/subject—possess/verb" brings back seven sentences from the real notes and no sentences from the simulated notes. The seven sentences are:

note 4, sentence #21
SHE KEPT AFTER ME.

note 6, sentence #18
BUT GAITY YOU (WOMAN) SAVED FOR STRANGERS.

note 10, sentence #7
AND SHE WOULD STAY WITH ME.

note 23, sentence #7
TOO BAD YOU (WOMAN) JUST KEPT EVERYTHING INSIDE YOU (WOMAN).

note 24, sentence #28
YOU (WOMAN) HAVE EVEN MORE THAN YOU (WOMAN) HOPED FOR.

note 26, sentence #10
I HOPE YOU (WOMAN) HAVE ALL THE LUCK IN THE WORLD.

note 29, sentence #5
(YOU [WOMAN]) KEEP EVERYTHING QUIET AS POSSIBLE.

These retrieved sentences tend to picture women as being rather powerful, the ultimate deciders of what is or is not shared with, or given to, the writer.

Note in these retrieved sentences the importance of text editing. Part of the optional preparation procedure involves identifying proper names and ambiguous pronouns. Five of these seven sentences would have been missed if the sex of the second person pronoun were not identified. Syntax marking is used to separate the verb "have" as a main verb from "have" used as an auxiliary. Only "have" used as a main verb is tagged with the label "possess".

Many of the other sentences having "female-role" as subject are concerned with giving instructions in both the real and the simulated notes. In an analysis by Ogilvie et al.,[13] using our earlier dictionary, it was found that these requests in the real suicide notes were rather specific, such as, "YOU (WOMAN) TELL MY FOLKS," or "YOU (WOMAN) PLEASE TAKE CARE OF MY BILLS," while the requests in the simulated notes were much more vague, e.g. "YOU (WOMAN) FIND A NEW LIFE FOR YOURSELF."

Question retrievals often help to locate the cause of a particular tag count. In the left hand column of figure 4b, for example, we noticed that the tag "natural-world" was high for simulated notes in both subject and object positions. In looking at a retrieval printout of this category, we noticed that many of the tag counts for the simulated notes were caused by the text word "life", particularly as an object in conjunction with the tag "self" as subject. While life itself is a part of nature, it certainly differs from (and is admittedly somewhat misplaced with) other entry words in this category, such as those pertaining to weather and cosmic objects. A special retrieval was made for all those sentences that combined the tag "self" as subject with the appearance of the actual text word "life" as an object. Eleven sentences were retrieved from the text of simulated notes, only three from the real notes. Given the fact the real text is considerably longer, this is quite contrary to chance expectation. The retrievals are well distributed over the entire text. Both real and simulated note retrievals yield messages quite similar in content: "I CANNOT FIND MY PLACE IN LIFE." "BUT I JUST CANNOT STAND LIFE ANY LONGER." "YOU SEE, I KNOW NOW THAT I CAN NEVER HOPE TO REALLY MAKE A SUCCESS OF LIFE." "BUT I CAN NOT SEEM TO STAND LIFE THIS WAY." "I HAVE NOT MADE LIFE SEEM WORTHWHILE." "AND I CAN SEE NO USE IN PROLONGING IT (LIFE)." "IN A FEW MINUTES I WILL TAKE MY LIFE," etc. All but the first of these sentences are from the simulated notes.

The results presented here are meant only as examples of the kinds of analyses that can be done using General Inquirer operations. The tag categories listed in figure 4 are not the only ones to be used in making analyses. Two categories, each of which may not differentiate the two texts by itself, may very well serve to differentiate when combined in a retrieval question. We have already written two articles[13, 14] describing our work with these notes. A third article probably will be written this summer. Suffice it to say here, in summary, that the notes can be differentiated by the General Inquirer.

Our analyses find real suicide notes characterized by references to concrete events and interpersonal relationships (the role of women as powerful and denying being of particular significance), and simulated notes characterized by their abstract intellectualization, often amounting to a reflection on the relative merits of life and death.

AUTOMATIC THEME ANALYSIS

One of the problems frequently facing the General Inquirer user is the construction of question sets that tap more than a small fraction of the sentences in the text. For example, an investigator may be able to develop questions that are very powerful in discriminating sentences of source *A* from those of source *B*, but the number of sentences used in making such discriminations may involve a total of as little as twenty per cent of the entire text. What about the rest of the text? May it not also have some discriminating features?

One possibility is that the remaining text does contain features that would permit discrimination between the two sources, but that the investigator must ask different questions in order to find them. Perhaps the investigator's theory is orthogonal to the real differences lying within the texts. Another possibility is that there are real differences between the texts, but that the dictionary itself is not relevant. If this is the case, then there is no better alternative than to develop better dictionaries. Finally, there remains the possibility that there are no differences between the two texts. If this is the case, we are stuck. However, as we shall see, the actual state of "no difference at all" in such a multiple descriptive instrument as the General Inquirer is quite rare.

From our experience in using the General Inquirer, we came to feel that the main difficulty in developing rules that would discriminate a greater percentage of the text was due to our rule building procedures, not to any inadequacies of our dictionaries or the lack of real differences in the text itself. Dr. Hunt, who was then at Yale, suggested that perhaps we could use computer concept formation procedures to help find more discriminating rules. He offered his own Concept Learner[3] as a computer system for putting this into effect. Probably many persons in this audience remember hearing the Hunt-Hovland predecessor of this system described at the 1961 Western Joint Computer Conference.[2] The current Concept Learner (CL-2) is very large and has application to a wide variety of problems. There was no doubt that CL-2 could do at least an "interesting" job of building rules for discriminating between two text sources.

One source of considerable humor in the last few years has been the characterization of computers as having semi-human qualities. While the idea of two computing machines marrying and having baby computers is perhaps best left to fantasy, the merging of two large scale computer program systems to produce a new program system with an identity of its own is indeed within the realm of possibility. What follows is the story of such a romance.

As with any large scale affair, there were some technical difficulties. In this case, our two principals, the General Inquirer and the Concept Learner, did not speak the same language. The General Inquirer is programmed entirely in COMIT, a computer language developed by Yngve and his mechanical translation group at M.I.T.[15] The Concept Learner is programmed entirely in IPL-V, a computer language that came from Carnegie Tech and the RAND Corporation.[16] Fortunately, the General Inquirer and the Concept Learner had two bi-lingual friends, namely Hunt and myself (Stone), to act as interpreters. A programmed interlingua was developed for handling translation problems and the merged operation was on its way.

The resulting procedure is this: the General Inquirer gives to the Concept Learner (via our interlingua) a list of tags that have been applied to each sentence of both texts *A* and *B*. While the original text words are dropped, care is taken to note the syntactic position that was associated with each tagging operation. Each sentence is thus replaced with a list of syntax marked tags (hereafter called "labels"). Given sentences described in this form, the task for the Concept Learner is to find a near minimum set of "rules" for distinguishing as many sentences as possible of document *A* from those of document *B*.

Rather than have to consider all the complexities and interrelationships of actual language, the Concept Learner is provided by the General Inquirer with a description of each

sentence consisting of a list of tags chosen from the 83 different categories, each tag marked with one of the five different syntax positions. There is thus a finite resource of 415 different labels from which the dozen or so symbols used in describing any particular sentence must be selected. The number of combinations of 415 labels, taken a dozen or so at a time, is very large, especially when any one label may be used more than once in a sentence. It is not surprising that we find it very rare for two or more sentences to have identical descriptive lists. So long as there remain differences in these descriptive lists, we have grist for building discriminative functions.

The Concept Learner looks at all the sentences in each document to see if there are one or more labels common to all sentences in one document that are not found in any sentences of the other document. For example, if *all* the sentences in document A were retrieved by these two labels:

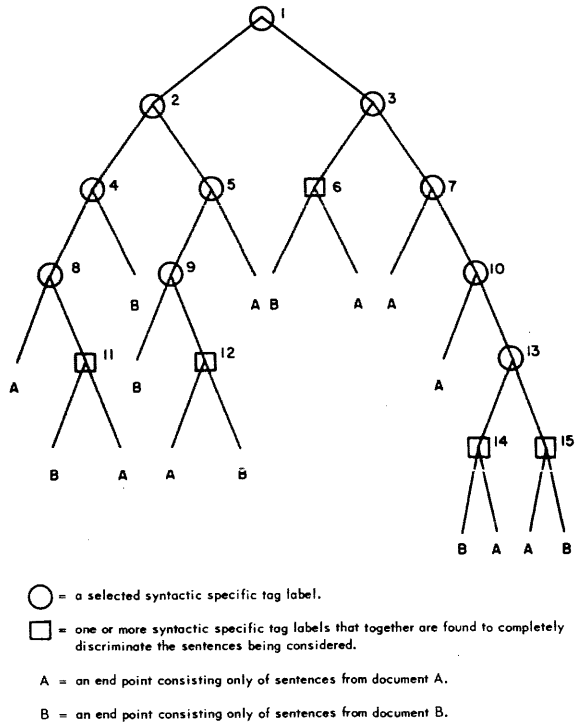
1) The tag "family" in the syntactic position of subject.
2) The tag "control" in the syntactic position of verb.

but *none* of the sentences in document B matched both these specifications, then this single question, by itself, could be considered as completely discriminating.

Usually no single question can successfully serve as an "all-none" test for discriminating all the sentences in one document from all the sentences in another. Thus, it becomes necessary to develop a discriminating procedure that uses a number of different questions. The tree procedure is one means for doing this.

The basic tree building strategy in CL-2 is based on the following heuristic: if there is not a single label or combination of labels that completely discriminates the sentences in document A from those in document B, the Concept Learner takes whichever document has the fewest sentences and finds which label occurs in the most of these sentences. The occurrence of this label is then used as a test to divide the sentences in documents A and B into two subsets each, depending on whether the label is present (sets A_{1+}, B_{1+}) or absent (sets A_{1-}, B_{1-}). The search for a successful all-none discriminating test can now be applied separately to each of these new subgroup pairs. First, subgroup pair A_{1+}, B_{1+} is tested. If a successful all-none discrimination is made on this subgroup, the computer can immediately turn its attention to finding a successful all-none test for subgroup A_{1-}, B_{1-}. If no successful all-none test is found, the subdivision process continues until either a sub-subgroup is found where an all-none test does apply or one of the document sources runs out of sentences.

Figure 5 shows an example of this subdivision process. A branch to the left represents the sentences that successfully matched the specifications at this node. A branch to the right represents the sentences that failed to match this specification. In actual operation, the tree keeps subdividing to the left until a complete discrimination is made. It then works on the A_{n-}, B_{n-} subgroups, starting from the bottom subgroup (i.e. the right branch of node #8) and working back up.



Some results using the Concept Learner on text analysis problems. There are several criteria that are useful for evaluating the success of the Concept Learner. One is the simplicity of the tree. Obviously, if the total number of nodes in the tree almost equals the number of sentences in both documents combined, we have

not done much towards finding trends in our data. A second test is how well a developed tree can identify correctly the source of new sentences. To test this, we divide our data approximately in half (as we did in the suicide note study) and use only half the data to develop a tree. The tree is then used to classify the other half of the data, the percentage of correct classifications being our evaluation index.

So far, the Concept Learner has been used on about a half dozen automatic theme analysis problems. Identifying reliable differences where each source is the work of just one author turns out to be much easier than identifying reliable differences where each source is the work of a group of authors. Most of the trees developed on multiple author sources have been quite complicated, some containing upwards of sixty test nodes. Usually, these more complicated trees are not very good at classifying additional test data correctly.

The suicide note problem, of course, is one where each source contains multiple authors. Partly because of this, the Concept Learner yields a rather complicated tree that has less success in discriminating real from simulated notes than our regular man-machine General Inquirer procedures reported above. In one respect, this lack of complete success with all machine procedures is somewhat of a relief. After all, we do not want to create technological unemployment for ourselves as psychologists.

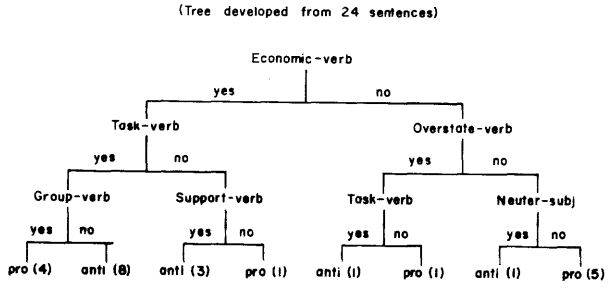
As a less complex example, let us consider a study involving limited amounts of data with only one author to each source. The data here consists of four documents circulated in connection with the 1962 California state election.

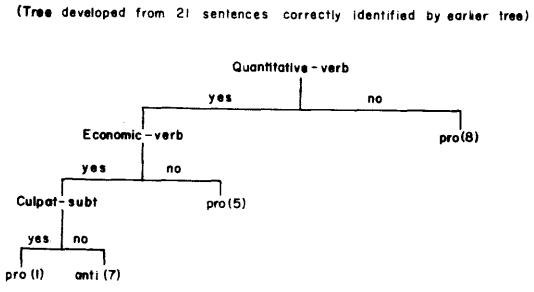
California is famous for requiring its citizens to act as legislators. The typical California election ballot presents the voter with twenty to thirty quite complex issues. To aid him in decision making, the Secretary of State distributes a booklet describing the various propositions on the ballot. This booklet contains brief arguments by proponents and opponents of each measure.

In 1962, an unusual event occurred. Substantially the same proposal appeared on the ballot twice. Proposition 1, which arose in the State Assembly, was a proposal which would have authorized an increase in the pay of state legislators to $11,000 a year. Proposition 17 was a proposal by the State Senate to raise salaries to $10,500. Different paragraphs were included in the booklet urging and opposing each measure. The arguments were quite close in each case. Both opposing arguments were written by a California property owners association, the supporting arguments by committees of the legislature. (As might be expected, the voters apparently perceived the similarity between the two propositions; both measures were defeated at the polls.)

Twenty-four sentences were selected from the arguments for and against Proposition 1, and twenty-eight sentences from the arguments for and against Proposition 17. Sentences which obviously indicated the source, such as "Vote no . . .," were excluded.

The sentences for and against Proposition 1 were processed through the General Inquirer and then analyzed by the Concept Learner. The result of the analysis is the tree shown in figure 6a. The original twenty-four sentences break down into eight groups, each group corresponding to an end point in the tree. Four of these groups collectively contain all the pro statements; the other four groups contain all the opposing statements. Thus, if the inquiry is: "economic/verb, work/verb, and group/verb,"





there will be only four sentences in the text that match all these specifications, and they will all be in support of the proposition. However, the discrimination rule "economic/verb, work/verb, and *not* group/verb" refers to a set of eight sentences, all of them against the issue. From the point of view of making General Inquirer retrievals, the entire tree structure can be converted directly into "question sets"; each branch of the tree defines a separate question.

Clearly, the amount of data reduction in figure 6a is insufficient. Four of the eight end points refer to only one sentence. These probably represent specialized questions that are very unlikely to be of discriminative value in future analyses. Probably more powerful trees can be developed to represent the differentiating main trends in the data. Note, for example, that "economic/verb" by itself is able to distinguish eleven of the thirteen anti sentences.

Nevertheless, this tree does fairly well when used to classify the twenty-eight sentences from Proposition 17. Twenty-one of the twenty-eight sentences were classified correctly. If these twenty-one correctly classified sentences are given to the Concept Learner, a new, much simpler tree can be grown, as shown in figure 6b. Basically, the sentences against the propositions are concerned with "economic/verb, quantitative/verb" aspects of the issue. Non-economic quantity references are in favor of the issue, as are sentences not mentioning quantity references in the verb at all. Those arguing against the bill are apparently preoccupied with its economic costs; those supporting the bill tend to focus on its other features. Granted, most California citizens could have told us this without our having to go to so much trouble. Yet our machine has done thus via explicit procedures, where our average citizen might be a bit hard pressed to explain his exact cognitive reasoning. Maybe we can use one to help understand the other.

As we have seen, the Concept Learner builds trees using two main heuristics: an "all-none" test for terminal nodes and special non-terminal procedures for dividing text into subsets. There are many other procedures that might also be explored. We have decided to expand the Concept Learner to handle these other possibilities. Given the inordinate proportions of computer time consumed by our interlingua routines, it was decided to use this opportunity to reprogram the Concept Learner, plus the new procedures, in the COMIT computer language. COMIT has proven to be well suited to the task. The program has been designed and written at Harvard by Mr. Marshall Smith and is currently in the debugging stage. Hopefully, it will be running at the time of this meeting.

The principal difference between the new program and CL-2 is the number of tests available at a given node. By using flexible dispatching procedures, the investigator can select tests from the following list and order them in any way he wants. All tests offer minimum cutoff parameters which are again adjustable by the investigator. The list of tests is as follows:

1. *All-none test based on a single label.* Looks for a label that characterizes all the sentences in source *A* but none in source *B*. This failing, it looks for a label occurring in all the sentences of source *B*, but none in source *A*.
2. *Some-none test based on a single label.* Looks for the label that both occurs in the largest number of sentences in source *A* that is not represented in source *B*. A similar search is made of labels in source *B* that do not occur in source *A*.
3. *Many-few test based on a single label.* A search is made for that label which has the largest absolute difference in the number of occurrences in source *A* versus source *B*. If the number of "few" sentences is below a certain minimum specified by the investigator, they are thrown away and the left hand side of the node is marked as being terminal. If the number of "few" sentences is above this minimum, the left hand side is considered non-terminal, and further subdivision occurs.
4. *Hunt non-terminal heuristic.* The computer determines which source contains the fewest sentences at this node and finds the label that occurs in the largest number of these sentences.

By basing all our tests on a single label, it is possible to execute tests quickly, using matrix procedures rather than list commonality searches. The cost is that we cannot then develop single node discriminations based on

a conjunctive combination of labels. While this is essential for other Concept Learner problems, it is not necessary for automatic theme analysis when some-none or many-few tests are also provided.

One addition planned for the near future is a "look ahead and evaluate the various possible outcomes" routine that will allow the investigator to direct the tree building procedures according to his own theoretical interests. Since it would be inconceivable for the machine to search ahead and try out all possible outcomes, it will look for a list of suggestions supplied by the card reader. All these suggestions will then be evaluated by the computer in terms of the extent to which they lead to certain desirable tree properties. The most highly evaluated suggestions will then be used in building the tree.

FINAL REMARKS

While most of our research examples have been given mainly to illustrate procedures, we would not want to leave the impression that we are blind to the many possible General Inquirer applications. The General Inquirer is currently being employed by different investigators on a variety of theoretical and applied problems, both serious and sometimes somewhat amusing. Data already under study include small group discussion procedures, diplomatic notes exchanged between countries, personality differences between northern and southern negroes, congressional subcommittee testimony of different lobby organizations, delusional language of schizophrenics, college applications, descriptions of magico-religious role differentiation in primitive societies, differences in attitudes toward friendship in large and small town environments, peace corps field reports, comparison of police chiefs and probation officers regarding their attitudes toward the juvenile delinquent, the letters written by the "Three Faces of Eve" as a study in multiple personality, cross-national comparison of college students' plans and expectations for the future, and thematic changes in popular song lyrics during the last 30 years. While we cannot discuss so many studies in detail here, the results so far have been very encouraging. Pleased with our initial results, we hold optimistic expectations for the future.

REFERENCES

1. STONE, P. J., BALES, R. F., NAMENWIRTH, J. Z. and OGILVIE, D. M. The General Inquirer: a computer system for content analysis and retrieval based on the sentence as a unit of information. *Behavioral Science*, 7, 1962, 484-498.
2. HUNT, E. B. The development of decision trees in concept learning: model and basic results. Working paper number 6, Western Management Science Institute, U.C.L.A. April, 1962.
3. HUNT, E. B. *Concept Learning: an Information Processing Problem.* New York: Wiley, 1962.
4. STONE, P. J. and HUNT, E. B. The General Inquirer extended: automatic theme analysis using tree building procedures. *IFIP Proceedings,* Munich, 1962.
5. BERELSON, B. Content Analysis, in Lindzey, G. (ed.) *Handbook of Social Psychology.* Cambridge: Addison-Wesley, 1954.
6. MOSTELLER, F. and WALLACE, D. L. Inference in an authorship problem: a comparative study of discrimination methods applied to the authorship of the Federalist papers. Department of Statistics, Harvard, 1962.
7. MCPHERSON, W., DUNPHY, D., BALES, R. F., STONE, P. and OGILVIE, D. M. A Revised Psychological and Sociological Dictionary for the General Inquirer. Dittoed paper (two volumes) Laboratory of Social Relations, Harvard University, December, 1962.
8. KLUCKHOHN, C. The scientific study of values. University of Toronto Installation Lectures, 1958.
9. HOLSTI, O. R. Computer content analysis. Working papers numbers 1, 2, and 3. Stanford Studies in International Conflict and Integration, 1963.
10. OSGOOD, C. E., SUCI, G. J., and TANNENBAUM, P. H. *The Measurement of Meaning.* Urbana: The University of Illinois Press, 1957.
11. OSGOOD, C. E. The effects of motivation on style of encoding. In Sebeok, T. (ed.) *Style in Language.* New York: Wiley, 1960.
12. BALES, R. F. and STONE, P. J. A general psycho-sociological dictionary for the Gen-

eral Inquirer. Laboratory of Social Relations, Harvard University, 1961. Ditto (also briefly described in Stone and Bales, *et al.*)[1]

13. OGILVIE, D. M., DUNPHY, D. C., SMITH, C., STONE, P. J., with SHNEIDMAN, E., and FARBEROW, N. Some characteristics of genuine versus simulated suicide notes as analyzed by a computer system called the General Inquirer. Laboratory of Social Relations, Harvard University, August, 1962. Ditto.
14. STONE, P. J., OGILVIE, D. M., and DUNPHY, D. C. Distinguishing real from simulated suicide notes using General Inquirer procedures. Paper read at the joint annual meeting of the American College of Neuropsychopharmacology, Washington, D. C., January 25, 1963.
15. YNGVE, V. *COMIT Reference Manual.* Massachusetts Institute of Technology Press, 1962.
16. NEWELL, A. (ed.) *Information Processing Language V Manual.* New Jersey: Prentice Hall, 1962.

SELECTIVE DISSEMINATION OF INFORMATION (SDI): STATE OF THE ART IN MAY, 1963

C. B. Hensley
International Business Machines Corporation
Advanced Systems Development Division
Yorktown Heights, N. Y.

INTRODUCTION

Selective Dissemination of Information (SDI) is a new and rapidly developing field. The concept was originally set forth by Hans Peter Luhn in 1958.[1, 2] As described by Luhn, one part of a larger idea, the business intelligence system, was Selective Dissemination of Information. SDI involves the use of the computer to select from a flow of new documents, those of interest to each of a number of users. This process may be thought of as the inverse of information retrieval. In information retrieval, a user precipitates a search of a file of documents. In SDI a document precipitates the search of a standing file of user interests. SDI has been called "current awareness" since the attempt is to keep the user aware of current developments. This function has been traditional with those few really excellent librarians and executive staff assistants. SDI is a mechanization of this function.

The concept described by Mr. Luhn was first implemented in 1959. At that time in Yorktown Heights, New York, an IBM 650 Data Processing System, together with other card machines, reproduction equipment and human operators, processed a small flow of documents against the interest profiles of some 30-odd users. This system has subsequently been called "SDI 1."[3, 4, 15] In 1960 a second system, SDI 2, evolved from the original one. SDI 2 is the first system designed and documented so that it may be installed remotely.[5, 6] In 1961, documentation for the SDI 2 System was completed and the first public announcement of a documented SDI System was made on July 11. Implementation started on a third system, SDI 3, early in the year. Although SDI 3 was in partial operation during the last part of 1961, complete debug and documentation was not completed until 1962. During 1961, several other systems, all of which will be designated by their location names, became operational. These included SDI Owego,[7] at the IBM Federal Systems Laboratory in Owego, New York; Current Awareness Service, a System at the Technical Information Center at General Electric's Evandale installation,[8] and a manual current awareness service available at cost to all United States Citizens at the Office of Technical Services of the United States Department of Commerce in Washington, D. C.

More systems became operational in 1962. The second system tested and documented for remote installation, SDI 3, became available through the IBM Data Processing Library for the IBM 1401 Data Processing System.[9] The Poughkeepsie System went into operation near mid-year at the Technical Information Center of the IBM Data Systems Division in Poughkeepsie, New York.[10] A fourth system from the Mohansic Group,* SDI 4, has been in partial operation for nine months and documentation

* Mohansic is the IBM ASDD laboratory in Yorktown Heights, N. Y., where SDI 1-4 have been developed and tested.

is well under way.[11] The Douglas Aircraft Corporation in Santa Monica, California, has a system in an advanced state of debug.[12, 13] Around the end of 1962, a second 1401 program became operational at IBM, Data Processing Division, Midwest Region, in Chicago, Illinois. Over the past four years, SDI has moved from a concept into a rapidly increasing number of system installations.

Implementation

Implementation difficulties for a system are often underestimated. This is in contrast to reduced operational cost and increased quality of service which are often overestimated. With SDI, the choice is whether (1) to use an available system, (2) modify an available system, or (3) to write your own new one. Because of the uncertainty, implementation cost is hard to estimate. Quality is even more difficult to estimate. Everyone seems to feel he is an expert on quality. There is disagreement in many cases. Present SDI systems involve computer programs, manual procedures and sometimes special equipment. In order of increasing difficulty, implementation may involve the installation of a well-documented, tried and true system which is in operation somewhere else; modification of manual procedures; obtaining special equipment; reprogramming or redesign.

Human skills available; experience of the personnel with SDI, Information Retrieval or related areas; and the number of other systems, procedures or constraints interacting with the new SDI installation all affect the effort required for implementation. Not only are a wide systems background, computer knowledge, and documentation experience valuable, but specialized knowledge with office machinery, industrial engineering, typography as well as psychology, sociology and organization theory often help. Programmers seem to be necessary for any type of installation. The more experience with data processing as contrasted with scientific programming the better, but any programming experience is better than none. Systems and procedures personnel are well-known in most organizations and are certainly advantageous for modifications or rewriting.

Experience with installations of documented SDI systems is limited. It was estimated that three calendar months and a total of three man months effort would be necessary to install an early SDI system.[3] Programmers have been used in all cases. The time to get a SDI program through a monitor system or to fit in with other existing operating procedures has been quite variable. In one case the program assumed a particular load routine long in general use, but not in use in this installation.

Experience with combining and modifying existing systems is exemplified by Poughkeepsie. There, despite the fact that the programs had little, if any, documentation, one or two programmers fought through SDI, KWIC* and an IR program in a few months. The manual procedures were in flux for a longer period. The total system is still being modified and only parts are in operation. Owego was a rewrite from SDI 2 which took over a calendar year to get into operation. The program rewrite itself, from start to run, took about three months. Prestart systems work extended longer and, to my knowledge, the system is still rather weakly documented for remote installation[18, 19] and is being integrated with KWIC.[7] What might seem to be a relatively simple rewrite of SDI 2, SDI 3, required one person for a calendar year in a building being noisily rebuilt, although the programming and documentation was done by an experienced programmer who knew SDI and the machine.

The classic problem seems to be an underestimation of the amount of the programming required to rewrite and document. For experienced personnel, e.g. SDI 3, estimates seem to be low by a factor of four. For less experienced (with SDI) personnel perhaps six would be better, e.g., Poughkeepsie. It should be pointed out that certain phases can sometimes be estimated accurately, e.g., programming at Owego.

User Interests

Most user profiles (interests) have been obtained without any problem by blindly mailing a short form to the potential user.† In three tests‡ some 65% of those contacted became users. Mass meetings of potential users have been used as well as blindly mailing longer

† Key Words in Context, a machine prepared printed index.[17]

* [5], Pages 94-5.

‡ [3], Page 41.

forms with either term dictionaries attached, e.g., Owego (modified ASTIA), or enclosing examples*** of indexed document items. Indirect methods have also been used to derive profiles from personnel or project information.*** Only with SDI 1 was a comparative study made and it had too small a sample to be conclusive.*** Each of these methods have been proven feasible. Further research is needed to define situations where one is preferable to another.

Adjustment of user profiles has been done largely at the user's instigation. At Mohansic, blanket mailings of current user profiles with change forms have been made to encourage users to make changes. Users have also been notified that they can make changes. The effectiveness of these measures is subject to doubt. The only known attempt to automatically update or adjust profiles based on user's responses was tried at Mohansic on SDI 1. The results were inconclusive. Manual attempts to suggest or arbitrarily make changes in user profiles based on various hypotheses have been made from time to time, usually without controls.

Although how to get new users to join and give the "best" possible profile seems to be a difficult theoretical problem, in practice there seems to be no difficulty. Experiments with automatic updating are in order but adequate user response histories seem to be necessary.

The number of users serviced by SDI systems now in operation has ranged from tens, to one to two thousands. Experience with larger groups is lacking although no new problems are anticipated. One problem, not initially anticipated, which increasing number of users has proven to be important, is that of address changes. These occur so frequently that not only must they be considered part of every normal run, but provision is necessary to change addresses between notification and hard copy order. As we shall see below (Abstracts and Notifications) this affects the notification itself.

Documents

Document sources for SDI are usually defined by the application. The range of subject matter on which there is experience is quite wide, including science, engineering and management. There are no known cases of letters, memorandums, or picture annotations being processed although this has been proposed and no problems are anticipated. Document source has been shown to be a significant factor in response.[14] Owego uses ASTIA documents predominately. Poughkeepsie uses internal IBM reports. Surveys of what users read[15] or library usage could be used to determine what document sources to use for an SDI system. Most such data indicates a skew distribution of usage with a few highly used journals. It is assumed but not demonstrated that different types of users need different document sources. Experimentation in this area might influence the selection by professional journals of items to abstract. SDI provides a tool in this area through its response. SDI 4 and a revised Chicago system will allow exclusion of documents by source, e.g. need-to-know or excluding journals user subscribes to. Volumes of document items being processed in SDI systems run from tens to hundreds per day with experience upward lacking. Subscribing to a journal is not much of a problem, but getting on internal distribution lists is more difficult than one might expect. It cost the Mohansic group several man months of effort to locate internal sources of information and arrange to be added to these distribution lists.

Documents normally come to one location, are handled and numbered. Some SDI's integrate with library operations to various degrees. Owego uses the same numbering and hard copy reproduction procedures. Mohansic provides abstract sets and utilizes journals from the library. Douglas is partially integrated. Some work with IR systems, e.g., Evandale, Owego, Washington. Document numbering may be sequential as at Mohansic or by an internal code as at Owego and Poughkeepsie. Checking for duplication and series completeness is a normal library problem.

There seem to be few serious operational problems in this area. Studies are needed to test automatic procedures to analyze user responses and to vary the document source mix to maximize value functions. Little has been done to study the effect of frequency of mailings to the user.

*** Ibid., Page 6.

Indexing and Decision

The primary decision method used in SDI has been a probability of interest estimate, i/d, where i is the number of words identical in the two lists and d is the number in the document list.* The words are normally chosen by humans** from text and truncated to adjust for endings. This particular technique came from a programmer's misunderstanding in 1958 of H. P. Luhn's instructions. It proved to work and was therefore kept. The proposition is that probability of interest increases with i/d. The i/d criterion may vary by document—Mohansic —or by user—Owego, Poughkeepsie and Chicago.[20] A no-truncation fixed-dictionary system with a thesaurus (Owego, ASTIA thesaurus) has been used with i/d. Experiments have been run at Mohansic, but not as yet reported, on truncation, 4-9 characters;[9] and depth of indexing, 1-26 keywords; as well as machine indexing from partial text by several methods. Conventional "Boolean" methods with keywords are used at Evandale. Chicago indexes by machine from the abstract using KWIC methods, i.e. dropping common words—A combination i/d and "Boolean" method with variable truncation is used.

A variety of indexing and decision procedures have been used. There needs to be work to compare the results under varying conditions. Relationships between SDI and IR need to be explored empirically in the indexing and decision area. Are they the same or different; if different, in what ways? More work needs to be done on the desirable amount of direct user control over the decision.

Abstracts and Notifications

The decision is made to notify the user of one or more items. What should the notification consist of? In one of the SDI tests, hard copies were sent directly to the user.† Users preferred a two-stage over a one-stage procedure: receive abstract notifications and be able to order hard copy instead of receiving the hard copy directly without intermediate control.‡ However, their reading habits seem to be considerably more effected by direct documents.*** With this exception, SDI systems have used two-stage procedures sending abstracts to users and allowing them to obtain hard copies, in some cases providing order forms. Abstracts have been compared to titles and the titles seem adequate for deciding which document to order, but abstracts are necessary partially to substitute for the document if the document is not available.[14] Quality of abstracts has been discussed in theory. There is no known experimentation. SDI and IR, with appropriate response evaluation, would appear to be excellent vehicles for this exploration.

A number of forms for notifications have been discussed and tried. IBM cards have been the most frequent vehicles. The abstract is normally typed on a reproduction master and reproduced onto cards. Normal card stock works well with offset or stencil but spirit master runs are too short. Chicago and Owego use or will use the IBM 1403 Printer to print on continuous-form abstract cards directly. A machine record of the form number and the user and document would have to be kept so that when the card was returned the response could be mark sensed and the number read. Mark sense requires special pencils which deposit an electrically conductive mark, but new optical machines (e.g., the IBM 1418 Optical Character Reader and IBM 1428 Alphameric Optical Reader), allow standard No. 2 and No. 3 pencil marks. Machine (1403) printing of the form number in place of prepunches is possible with an odd font. These have yet to be tried for SDI but appear cheaper for high volumes.

Single (SDI 1) vs. multiple (all other) card systems have been under debate since 1959. This debate no doubt will continue. The notification should combine (1) the document abstract (preferably both 3 x 5″ and IBM card size), (2) the user's address (3 or 4 lines for complete postal address which constantly changes), (3) the system return address, (4) questions regarding the document, (5) provision for the user's remotely made response to the questions, (6) the document number and (7) the user identification. The notifications

* See[5], Page 30;[3], Page 9, 10;[4], Page 8, 10.

** Education level doesn't seem to make any difference. See[16].

† See[3]; [4], Page 6.

‡ See;[3] [4], Page 10;[15], Pages 8-9.

*** See [4], Table IV.

should be in appropriate sequential order for mailing. If 5, 6 and 7 are not machinable on return, response handling for document hard copy orders and operating statistics must be manual, as in SDI 1. The abstract, 1, should be retainable by the user. A study[15] in one organization shows 3 x 5 and IBM card sizes were the most frequently used media for this purpose even prior to SDI. The response, 5, is made at many remote uncontrollable locations.

The PORT-A-PUNCH® card has proven to provide a machine readable response. PORT-A-PUNCH is only now (February 15, 1963) becoming available in continuous forms, thus making machine (1403) printing of the abstracts on the PORT-A-PUNCH card or an attached form possible. Previously a bill feed attachment was necessary which slows the printer. Systems remain to be developed and tested based on bill feeds, optical reading and many other devices. When several notices go to each user at once, placing several cards together (or using a sheet of paper as at Evandale) might save handling expense and user exasperation. No existing system meets all of these requirements; each compromises to some extent. Considerable research is necessary before sufficient basic knowledge is obtained as to the relative worth of these various features.

Response, Reports and Hard Copy

SDI 2-4 require the user to respond on every notice. Other systems require responses under certain conditions (SDI 1, no response if negative) or never, i.e., just a notification. It is not known exactly what effects this has.

Responses and other records allow reports to the user, operators, management and research personnel. This is a largely undeveloped area even though some rudimentary reports are included in the SDI 2 and 3 systems. Feedback reports could be used to assist in updating user profiles, changing the document sources mix, adjusting the system sizes, changing indexing methods, and adjusting the cost vs. value balance. Randomly selected notices (SDI 1-4) allow the system selection performance to be compared to random selection as a base. This also allows miss items (which could have been selected by the system but were not) to be estimated statistically.

There have been various hard copy procedures. (1) Ignore the problem (SDI 2-4). (2) Refer the user to a library (SDI 2-4). (3) Shelve and pull (SDI 1-4). (4) Keep vellum and reproduce (SDI 2-4). (5) Use aperture cards and reproduce (initially at Owego). (6) Use reel microfilm at multiple locations (Poughkeepsie). Adequate analysis of cost and value are yet to be made. Most systems agree with the Mohansic survey,[15] users want to be able to obtain hard copy.

Value-Cost

This is, in my opinion, the area with the largest potential for development. Available cost data[3] is very limited and hard to interpret. Available value information[15] is largely subjective. Dichotomy scales have been used in SDI, i.e., "of interest vs. not of interest." It is my opinion that ordinal, and cardinal scales are needed if we hope to move SDI design from an art towards a science.

REFERENCES

1. "A Business Intelligence System," H. P. LUHN, IBM Journal of R&D, 2, 4, 314-319, October 1958.
2. "Selective Dissemination of New Scientific Information with the Aid of Electronic Processing Equipment," H. P. LUHN, American Documentation, 12, 2, 131-138, April 1961.
3. "Selective Dissemination—Report on a Pilot Study—SDI 1 System," C. B. HENSLEY, T. R. SAVAGE, A. J. SOWARBY, and A. RESNICK, IBM, ASDD, Yorktown Heights, N. Y. Report 17-039, January 1961. (Presented at the 18th meeting of the Operations Research Society of America—1960), 45 pp.
4. "Selective Dissemination of Information—A New Approach to Effective Communication," C. B. HENSLEY, T. R. SAVAGE, A. J. SOWARBY, and A. RESNICK, IRE Transactions on Engineering Management, EM-9, 2, 55-65, June 1962, 11 pp.
5. "Selective Dissemination of Information—SDI 2 System," W. BRANDENBERG, H. C. FALLON, C. B. HENSLEY, T. R. SAVAGE, and A. J. SOWARBY, IBM, ASDD, Yorktown Heights, N. Y. Report 17-031, April 1961, 102 pp.

6. "The Selective Dissemination of Information System—Present Operations and Future Application," A. J. SOWARBY, pp. 14-40, in "Library Seminar—October 19-20, 1960" C. F. Balz, Editor, IBM, FSD, Space Guidance Center, Owego, N. Y., February 28, 1961.
7. "The Merge System of Information Dissemination, Retrieval and Indexing Using the IBM 7090 DPS," R. H. STANWOOD, IBM, FSD, Owego, N. Y., Report 62-825-441, April 1962 (presented at the September 1962 ACM meeting), 10 pp.
8. "New Concepts in Technical Information Services," B. K. DENNIS in Proceedings of The Engineering Information Symposium, 19-23, NYC, January 17, 1962, available at $2.00 from the Engineers Joint Council, 345 East 47th Street, N. Y. 17, New York.
9. "SDI 3 for the IBM 1401 Data Processing System 10.3.004; Selective Dissemination of Information (SDI) for the 1401 Tape System, the 620 Tape System and FORTRAN II," A. J. SOWARBY, W. BRANDENBERG, H. C. FALLON, C. B. HENSLEY, and T. R. SAVAGE (IBM, ASDD, Yorktown Heights, N. Y.), 1401 General Program Library released June 25, 1962, 157 pp.
10. "A Computer Integrated System for Centralized Information Dissemination Storage and Retrieval," R. J. TRITSHLER, IBM, DSD, TIC, Poughkeepsie, N. Y. (Presented at the ASLIB Conference, Blackpool, Lancashire, England, October 4, 1962).
11. A. J. SOWARBY, IBM, ASDD, Yorktown Heights, N. Y., unpublished material.
12. "Mechanized Information Retrieval System for Douglas Aircraft Company, Inc., Status Report," SM-39167, Missile & Space Systems Division, Douglas Aircraft Company, Inc., Santa Monica, California, January 1962.
13. "Library Information Retrieval Program," G. W. KORIAGIN, Missiles and Space Systems Engineering, Douglas Aircraft Company, Inc., Santa Monica, California, Engineering Paper No. 1269, March 1962 (Presented to the American Chemical Society, Washington, D. C.) February 1962, 23 pp.
14. "Relative Effectiveness of Document Titles and Abstracts for Determining Relevance of Documents," A. RESNICK, Science, 134, 3484, 1004-1006, October 6, 1961.
15. "The Use of Diary and Interview Techniques in Evaluating a System for Disseminating Technical Information," A. RESNICK and C. B. HENSLEY, IBM, ASDD, Yorktown Heights, N. Y., Report 17-055, December 1961, 86 pp., scheduled to appear in the April 1963 issue of American Documentation.
16. "Comparative Effect of Different Education Levels on Indexing in a Selective Dissemination System," A. RESNICK, IBM, ASDD, Yorktown Heights, N. Y., Report 17-092, August 1, 1962, 16 pp.
17. "Keyword-in-Context Index for Technical Literature (KWIC Index)," H. P. LUHN, IBM, ASDD, Yorktown Heights, N. Y., Report RC-127, 1959.
18. "Selective Dissemination of Information," an IBM 7090 Program, R. BENJAMIN, S. D. MILLER and E. S. ROWLAND, IBM, FSD, Owego, N. Y. December 20, 1961 (in SHARE Library), 12 pp.
19. R. BENJAMIN, S. D. MILLER and E. SCOFIELD, IBM, FSD, Owego, N. Y., unpublished material.
20. "On Relevance, Probabilistic Indexing and Information Retrieval," M. E. MARON and J. L. KUHNS, Journal of the Association for Computing Machinery 7, 3, 216-44, 1960.

COMPUTER CONTROLLED PRINTING

M. P. Barnett, D. J. Moss, D. A. Luce and K. L. Kelley
Cooperative Computing Laboratory
Massachusetts Institute of Technology

I. INTRODUCTION

This paper describes some of the characteristics and applications of programs that have been developed recently in the author's laboratory, for the production of coded paper tapes to control the Photon photocomposing machine. Conventional typesetting machines have been supplemented in the last two decades by a variety of photocomposing machines that produce the original copy for photolithographic reproduction by a photographic process. In this process, images of letters and characters are focused by an optical system at appropriate positions of a roll of sensitized paper. A photocomposing machine contains a matrix of transparent characters in an opaque background, with a mechanism for illuminating one selected character at a time. Several types of photocomposing machines have been designed and manufactured. The work that is reported here has used some Photon machines that are installed in Boston, and which are equipped with paper tape readers.

The present photocomposing programs enable Photon paper tapes to be punched off line from 709 output that is formed from input that was read from Hollerith cards or Flexowriter tape, or which was formed within the computer in binary-coded decimal form, by conversion from an internal number representation. This makes it possible to photocompose conventional computer results of a numerical nature, and to use the computer to organize verbal and other material in routine ways, for subsequent photocomposition. This organization may be the mechanical imposition of format requirements that would require elaborate manual typesetting, on material that is punched on a simple keyboard device, such as a Flexowriter, with a separate description of output format, or interspersed parenthetic comments to specify format changes. The organization may entail selection, extraction and sorting of items of information, and more elaborate operations of verbal processing. The terms selection and extraction are used with specialized meanings—the selection of records which satisfy certain criteria (such as the occurrence of certain words) and the extraction of specified portions of successive records (for example the extraction of author names from abstracts of journal articles).

The features of the Photon machine that affect the programs are described in section II. The programs produce Photon code from the starting material by two or more successive string transformations. The so-called ω-strings that are punched on paper tape to drive the Photon machine contain codes that identify letters, and also codes that specify the spacing between letters, and codes that change the status of various units of the machine. This ω-code may be produced from any one of several sources—punched cards, Flexowriter tape, internal computer conversion and so forth. A large part of the construction of the ω-code is concerned with the calculation and accommodation of spacing requirements. It is convenient to produce ω-coded material from a representation of the material to be photocomposed that identifies characters by an enumeration which

takes account of case (i.e. capital or not) and which contains "operation" codes at points in the text where changes of format are to occur. Source material may represent characters in several different codes—the material on Hollerith cards is usually assumed to be in a single case, whilst Flexowriter tapes carry case shift indicators. It may be expedient to include format codes within the text when it is punched on an input medium. It may be prohibitively tedious, however, for a human agent to do this, but trivial for a computer using simple scanning rules. For these reasons, it is of some convenience to use a second code, called ρ-code, as an intermediate form in which material to be photocomposed is expressed within the computer. The programming problems of computer controlled photocomposition are thus divided into two quite separate parts that have been developed in parallel, with a simple and clearly defined interface. The evolution of the programs that effect the ρ-to-ω conversion has consisted largely of the provision for an ever increasing variety of format requirements. The programs that produce ρ-code from Flexowriter, punched card and other sources have been concerned with a variety of specialized problems—special input routines for binary card images of Flexowriter and other paper tapes, accommodation of different Flexowriter codes, translation of mnemonic format requirements, correction and modification of input texts, expansion of composite format requirements (macros), separation of case in Hollerith texts, and cyclic insertion of format changes in tabular and itemized material. These matters give rise to numerous scanning problems, some of which are quite difficult, particularly for the last two of the topics just mentioned. For this reason, the scanning systems developed in the author's laboratory[1] now are being combined with the photocomposing programs, to allow an even greater variety of scanning problems, associated with the preprocessing of input, to be handled easily.

An early photocomposing program separated the overall problem into the preparation and processing of ρ-strings, but dealt with the latter matter by a single, relatively long FAP coded subroutine. This formed the basis of the so-called PC1 and PC2 systems, reported previously, and which were used in some initial experimental studies.[2] The programs to be reported here are heavily subroutinized, in a way that allows many special details to be accommodated in specific applications. It has been our objective to develop a general purpose program which could deal with a considerable variety of photocomposing problems that were defined in a suitable input language. A language and the corresponding program have been developed that take care of features which are common to many typesetting situations. Application of this program to a variety of practical problems, however, has brought to light several special situations that were not anticipated when the input language was defined. The fact that such situations might occur was recognized when the programs were defined, and the subroutinized structure allows ready adaptation of the system to these special circumstances. It is our intention to seek generalizations and common characteristics of these novel situations and to define convenient ways of expressing individual cases for inclusion in later input languages and accommodation by later general programs. This classification and categorization of printing situations perhaps is the most intriguing part of the work. The variety of matters with which the input language and programs can deal has already evolved through several stages of increasing complexity. It seems that hazarding an approximation to a useful language, encoding it and trying it on real applications is the most effective way to proceed. In this regard the authors have been very fortunate in the willingness of other groups at MIT, particularly the Department of Libraries, the Publications Department, the Technology Press and the Registrar's Office to provide test material, critical comment and technical advice that has supplemented the photocomposing problems of our own laboratory, and compensated for our ignorance of printing technology.

After the description of the Photon machine and the ω-code in Section II, the idea of interspersed format control is developed in Section III and is illustrated by a few examples produced from Flexowriter and punched card input that use the simpler operation codes of the control language. The general structure of the present photocomposing program is described in Section IV. The processing of input material in accordance with an output format that is specified separately is introduced in Section V by reference to some simple examples of nu-

merical and verbal tables punched on cards in a fixed input format. Some more complicated constructions of the control language then are described in Section VI. Some examples of the processing of input by reference to an output format that is specified separately, and which require fairly complicated scanning of the input then are given in Section VII. Section VIII contains some examples of photocomposed results of actual computations. The photocomposition of built-up formulae, and problems of page composition are discussed in Section IX. Verbal processing, such as editing, index preparation and so forth, is considered briefly in Section X in relation to the photocomposing work and some examples are given of photocomposed output of verbal processing programs.

It should be mentioned that the methods described here are applicable in some degree to other forms of tape-controlled photocomposing, typesetting and electronic display equipment, and that other groups are working on similar problems.

II. THE PHOTON MACHINE AND THE ω-CODE

The "optical stencils" of the Photon machine are provided on a glass disc, etched in eight concentric annuli of 180 characters each. Several hundred such discs are in existence, and further discs, etched with new combinations of letters and symbols may be produced in a simple factory operation which involves photographing cards on which the relevant characters have been drawn. An extensive library of such cards exists. The replacement of one disc by another is a trivial manual operation, comparable with the replacement of a control panel on a punched-card machine. When the Photon machine is in operation, the disc rotates at high speed about a fixed axis. During each revolution of the disc, a light beam may be projected along a fixed path, through one of the annuli on the disc that is etched with characters. The illumination persists for a time interval that is very short compared with the period of rotation of the disc. An almost static image of any character on the disc thus can be produced for a very short time interval during each rotation. The timing of the burst of illumination, within the basic machine cycle, and the choice of annulus thus constitute two coordinates, or components of an address, that determine which character is displayed during any rotation. The annulus is changed whenever necessary by a mechanical displacement of the axis of rotation. For many applications the switching between annuli is relatively infrequent.

The image of the character that is illuminated by the light beam is focused by a lens and prism system that determines the magnification of the character and also the position on the photosensitive material on which the image is focused.

The Photon machine with which we work can be driven by a paper tape that contains codes of several types. These include track specifications (a track is half an annulus of characters, and is identified by an integer in the range 1-16 that is called the disc level) and lens specifications (in effect scaling factors—the machines contain twelve different lenses that give type sizes in the range 5 point to 28 point) that appear on the tape only when they are to be initialized or changed. The tape also contains, for each of the characters to be printed, the sequence number (i.e. position around the disc) of that character, and the escapement (i.e. distance to be allowed between the left edge of that character and the next, measured in units of 2^{-8} cms). Further codes cause prism return (which has the same effect as a carriage return on a typewriter) and vertical spacing of the film. The disc level, lens size, sequence number, escapement, carriage return and vertical space codes constitute the ω-code, that is produced by the photocomposing programs and is written on magnetic tape, from which punched paper tape is produced off line, using an improvised magnetic to paper tape converter. The further details of the ω-code are given elsewhere.[3]

It may be mentioned that Photon machines hitherto have been operated by a keyboard which restricts the overall speed to that of keyboard operation. The work reported here allows the production of photocomposed material in a much faster and more economical manner.

III. THE INPUT CONTROL CODES—SOME SIMPLE EXAMPLES

The input to the photocomposing system consists of data—the text to be photocomposed—and what is in effect a program, written in a special language. This is the control language

of the photocomposing system. It contains a mixture of procedural and implicit instructions, and it has several chaarcteristics that are reminiscent of the more familiar programming languages which are in general use. The simplest way of using the photocomposing system, however, involves a mixing of data and "program" in a fashion that is not so common in high-speed computing. The word "program" is used here for the input instructions written by the user of the system and expressed in the photocomposing control language, just as a Fortran program is taken to mean a program written in the Fortran language rather than the Fortran compiler itself. Analogies can be sought for this mixing in other fields—stage instructions in a play, style comments and key changes in a musical score, control codes that set selectors in punched-card accounting—but very few in high-speed computing. Automatic machine tool control may provide some examples, and it may be that this "interspersed" programming, by parenthetic comment, will find wider use as computer control of other machines increases.

A Flexowriter may be used to prepare the source material for a photocomposing operation in the following manner. The text is typed on a standard Flexowriter, with the inclusion of control codes wherever necessary, contained within square brackets [ ]. This precludes the use of square brackets as characters of the input text, but square brackets can be obtained in the photocomposed output by a simple device that is described later. In the simplest instance, it is necessary to include one collection of control codes, between square brackets, at the beginning of the text, and a single control code [EN], at the end of the text. The control codes at the beginning of the text specify type style (i.e. disc level—there being up to 16 different styles of type on a single disc), type size (i.e. lens size), page width (i.e. "measure"), page length and any extra spacing that should occur between lines ("added lead"). Even these codes can be omitted, and a single initialize code [IN] given if a certain standard format is acceptable (unjustified 5 point Scotch with a 36 pica measure, zero added lead and 108 lines per page).

To obtain justified 8-point Scotch italic on a page 4 inches wide and 8 inches high, however, the codes [JU, LS8, DL9, LN288, PD576] would be given before the first word in the text. All dimensions in control codes are specified in points. A point is $\frac{1}{72}$ of an inch. A detailed explanation of these and other control codes is given elsewhere.[4]

Further codes may be included between square brackets anywhere in the text, to control the appearance of the material which follows. Most of these control individual aspects of the text's appearance which can be changed independently—for example type style can be changed without changing the size, and vice-versa.

In addition, macro-control codes may be defined at arbitrary points in the text. Each macro refers to an explicit set of control codes. Later reference to this group of control codes may be made by referring to the macro-code. The form of these macro-codes is described in Section VI.

Flexowriter input is treated as a continuous stream in which line breaks are imposed by the program, by reference to type size, page width, and justification considerations. Paragraph breaks in the input normally are preserved in the output, and indentations (but not trailing spaces at the end of a line) are preserved. All input spaces can be retained if necessary.

Material can be justified or kept unjustified by the use of appropriate control codes, and unjustified lines can be flushed left, centered or flushed right by the use of further codes. Justification is effected by expansion or compression of interword spaces, by reference to criteria that are specified by appropriate control codes.

New lines, new paragraphs and new pages can be started at points in a text that follow certain codes. Horizontal spaces and margin settings can be specified by further codes, as can different forms of vertical spacing.

Part of the Flexowriter input for a very simple application, and the corresponding photocomposed output of the earliest photocomposing program are included in Appendix I. This used a rather unwieldy convention for delimiting control codes (not the square brackets). The Flexowriter input and corresponding Photon output for another example of "straight matter" composed recently is included in Appendix II. An example of verbal

punched-card input, using a dollar sign as a case shift indicator, and corresponding Photon output is included in Appendix III. A more elaborate example of photocomposed text with corresponding Flexowriter input is provided in Appendix IV.

IV. PROGRAM STRUCTURE

A few aspects of the present photocomposing programs are now described. The programs were coded in Fortran II, with a few simple FAP coded input subroutines to read Flexowriter tape images from cards or magnetic tape. As mentioned in the Introduction, the various forms of input are processed by suitable programs that vary from problem to problem and which all produce material that is expressed in a so called ρ code, which is then converted to the ω code by a set of subroutines that has been used with only exceptional changes, for the applications that are reported here.

The conversion of the input described in the preceding Section into ρ-code is almost trivial. Each letter and symbol of the input text is represented in the ρ string by a positive integer in the range 1 to 90; different integers being used for the upper and lower case of the same character (when it can occur in two cases). An input space is represented by a zero in the ρ string, and operation codes are represented by negative Fortran integers. The input subroutines for Flexowriter material deal with backspacing and error correction codes in the tape, both in the representation of the text and within interspersed control information.

The conversion of ρ to ω code is effected by a hierarchy of subroutines that separate the several types of action that must follow, when control codes are encountered in the text, or when certain conditions are recognized by the various subroutines. Certain control codes set parameters for immediate, recurrent or delayed use as the "setting" proceeds. Such parameters include point size, disc level, page dimensions and so forth. Other control codes, and conditions detected by the program, require more extensive action: for example adjustments of interword spacing at the end of a line, and of interline spacings at the end of a page.

The programs that convert ρ code to ω code have a structure that may be called concentric. A start text subroutine STTEXT is called first to initialize page count and various parameters. A "ρ to ω text" subroutine RTOT is called next, and this subroutine retains overall control of the conversion until an [EN] (i.e., "end") code is detected (by a lower level subroutine), at which juncture an ascent is made through several subsidiaries of RTOT, and then out of RTOT, to its calling program. This then calls an end text subroutine NDTEXT to deal with certain bookkeeping details. If the ρ string is exhausted during the operation of RTOT or its subsidiaries, control is returned to the program which called RTOT, to obtain further ρ material, and then to re-enter RTOT and descend through its subsidiaries to the appropriate subroutine level. The "ρ to ω text" subroutine RTOT first calls a "start page" subroutine STPAGE. This then calls a "ρ to ω page" subroutine RTOP which retains control until an end of page condition arises, due to an end of page or end of text control code in the input, or accumulation of sufficient ω code to set a page. The end page subroutine NDPAGE is then called, and control returned to STPAGE to start another page, unless NDPAGE had been called as a consequence of an [EN] control code, when an appropriate return to, and exit from, RTOT results. The "ρ to ω page" subroutine RTOP calls a "start line" subroutine STLINE. This calls a "ρ to ω line" subroutine RTOL, and an "end line" subroutine NDLINE in turn. The action that follows execution of the NDLINE subroutine depends on whether it was called to end a line in the body of a page, or at the end of a page, or at the end of a text. The "ρ to ω line" subroutine RTOL has a similar structure, calling a "start section" subroutine STSECT, which in turn calls a "ρ to ω section" subroutine RTOS and an end section subroutine NDSECT. A section is a portion of a line that has independent margin settings, within which material may be justified, flushed left or right, or centered. The RTOS subroutine tests the successive integers that form the ρ string, and takes appropriate action depending on whether they are positive, zero, or negative. A positive integer, in the range 1 to 90, specifies a character to be set, and an escapement is computed from the "relative width" that is given to that character in the style of type that is being used, and the

point size, (and sometimes certain other typesetting parameters that need not be considered here). A zero in the ρ string indicates an input space, that is converted into an escapement by reference to the point size and the relative width that is given to an input space. A negative entry corresponds to a control code, and a further subroutine that deals with control codes is then called by RTOS. This is switched by the control code to a subroutine that deals with the particular code which has been encountered. This segregation of the effects of individual control codes facilitates continued expansion of the set of codes with a minimum of recompilation.

The nested structure of the program can be extended to include further layers, and this will be done to deal with problems of page composition.

It should be noted that the subroutines which end sections, lines and pages may modify material that has been formed in the ω string at an earlier juncture. Thus the end section subroutine can alter interword spaces if justification is to be performed. The end line subroutine establishes the vertical spacing that is recorded in the ω string before the codes for the first character of the line just set. At present, ω code is retained only for a line. Expansion of the program to deal with larger units of text in a coordinated manner (e.g. providing page justification, arranging "run arounds" to leave space for diagrams that will be visible when the relevant portion of the text is read) will require increased provision in the program for retrospective modification of the ω string that has been formed.

The status of the setting process at any instant is recorded by some 70 parameters. These are stored in a "status" array (actually in common storage). Whenever a parameter is changed, a record is kept in a "backtrack" array of the position in the status array of the parameter that has been changed, and of the superseded value. If overset occurs before an interword space is encountered, the contents of the backtrack array may be used to restore the status of the setting process to that which was current when the previous word-end was encountered. The backtrack array is cleared whenever the end of a word is reached without overset occurring.

V. EXTERNAL FORMAT SPECIFICATION

The examples of photocomposed material that were given earlier were produced from input that contained interspersed control codes. Some simple examples now are given of photocomposed material which was produced from input which did not contain interspersed codes, but which was processed by the computer to form a ρ string in which these codes were inserted in accordance with appropriate specifications.

Appendix V contains photocomposed material that was produced from a card deck by a program which specified a particular point size, type style, and page length, started a new line flushed left from a fixed margin, did not justify, and treated each blank column on the card as a space of a certain width.

Appendix VI contains photocomposed material that was produced from a bcd tape by a similar program, each printer record being set on a new line.

Appendix VII contains tabular material, set from punched cards, by a program which read certain format tables from control cards that preceded the data cards. The format tables contained a set of entries for each section of the output (i.e., portion that was set by reference to a given pair of margins). These entries consisted of a specification of the card field that contained the data to be set in the section, and the margin positions, vertical alignment (flushed left or right, centered or justified) and style and size of type for that section. The data from each card was set on a new line, and page length was specified on a control card that preceded the format table.

VI. FURTHER CONTROL CODES

A few further types of control codes that are used in input to the photocomposing program may be mentioned here. These are "macros," special character codes, and delayed effect codes.

It can be seen that in some types of application, certain combinations of control codes would occur repeatedly in the input, if interspersed coding were employed. For this reason, it is convenient to define macros, for an individual job, by suitable statements that precede a text in the input medium, and then to use the

names of the macros, between square brackets in the text, whenever necessary. At present 25 macros may be defined for concurrent use. A macro definition consists of the macro identifier $Mn(n = 1,25)$, followed by an = symbol and then the list of up to 12 individual control codes separated by commas that are to be incorporated. Each definition is enclosed in parentheses and the whole group of definitions is enclosed in square brackets.

The character set of input texts has been limited in the preceding discussion to the ordinary alphabet, numerals and punctuation marks. These normally occur on standard positions of the Photon disc. Special characters may be etched, however, on any one of the 1440 usable positions of a disc, and many discs contain a considerable number of mathematical and other symbols. Although these symbols are not represented on a Flexowriter or Hollerith keyboard, it is possible to assign arbitrary names to these characters and to define them in a similar manner to the macros. A character definition consists of a name of up to 6 lower case letters followed by an = symbol followed by the list of controls which give the requisite lens size and disc level change, if any, disc sequence number at that level and the disc level and lens size to which the text must return. A set of up to 12 characters may be defined in this manner, each definition being enclosed in parentheses and the set of definitions enclosed in square brackets.

When any special character is required in the text, it is obtained by giving the previously assigned name of the character, enclosed in square brackets. The control code sequence for that name will then be inserted in the rho-string automatically by the program.

A further type of control code, with which the programs will be able to deal shortly, will request some action to be taken after an appropriate delay. An example is a code which requests an inter-line spacing that is to take effect after the line in which it is encountered has been set. It would be convenient to allow mnemonic arguments for control codes that specify margin positions and other quantitative items of information, and to allow statements that specify progressions or cycles of values which these mnemonics should be given on the successive occasions that they are used. At present, control codes mostly take effect at points in the text which can be anticipated before the setting process is accomplished. Delayed action codes will make it possible to specify elaborate formats in which changes in style and so forth occur at points, such as line endings, which cannot be anticipated until the spacing considerations have been determined by the ρ to ω conversion. The use of mnemonic arguments that are changed on use is reminiscent of conventional indexing operations. The use of mnemonic arguments that are to be given values by the program which are consistent with the realization of some preset criteria, present further interesting possibilities of program design. The complexity of the control code combinations that are encountered in applications of the present program has already created a need for a higher level of input language to be converted into existing control codes by a suitable processor.

VII. INPUT SCANNING FOR FORMAT CHANGES

A few examples will now be given of material that was photocomposed from input which did not contain interspersed codes, and which required a nontrivial scan to determine where suitable control codes should be inserted by the program.

Appendix VIII contains part of a Union Listing of Chinese Scientific Periodicals that is being prepared by the MIT Libraries. The source material has been punched on a Flexowriter, with special symbols representing diacritical marks in an arbitrary correspondence. Certain types of input item are delimited by carriage returns, tabulations, slashes, periods and combinations of these and a few other simple criteria. The relevant control codes are inserted for new lines, indentations, columnar alignments, changes of type and diacritical marks, by a program that is specific to this application. Corresponding portions of Flexowriter input and Photon output are included in the Appendix VIII. The photocomposing subroutines have been modified slightly, so that when information for a title continues from one page to the next, a comment to this effect is set at the foot of the page and the relevant title repeated at the top of the next page. This required a modification of just three or four of

the subroutines and is one of the few changes that has been made to date to subroutines that are concerned with ρ to ω conversion for an individual application of the program. Decimal classification codes are dropped, and entry numbers introduced, by some further trivial changes in the program.

Appendix IX contains a page of a Bibliography of North American Geology, that was set from Flexowriter input with interspersed control codes as a test, at an early stage in the development of this work. The corresponding Flexowriter material is included for comparison. It can be seen that inserting the codes manually is tedious, and that some simpler input is essential for production work.

Appendix X contains a page of a Current Serials and Journals catalogue of the MIT Libraries, that was set from punched cards that contained no case shift indicators or bold face indicators. These indicators were inserted by a program which uses the Shadow subroutine[1] and some rules of capitalization that were formulated by R. W. Snyder of the MIT Department of Libraries. A definition table corresponding to these rules was used by the Shadow subroutine to construct a list of pointers to capital letters in the input text and to positions at which control codes were to be inserted to produce bold facing and tabulation in the production of the Photon tape. Similar methods, using Shadow, can be used to process many other examples of itemized material, such as catalogues or bibliographies, in which successive items are to be treated in a cycle which involves calling Shadow, and using the output which it produces.

VIII. COMPUTED OUTPUT

Two examples of results obtained by the computer, which were converted to Photon code without intermediate recording on conventional output media, are given in appendices XI and XII. Both were produced by the PC1 system, which includes a modified Fortran compiler which can deal with COMPOSE statements. These are output statements, comparable with PRINT and PUNCH, which contain the statement number of a FORMAT statement, and a list of variable names. The FORMAT statement may contain any conventional field specifications. It also may contain K fields each of which consists of a count of the symbols in that K field, then a letter K, and then a sequence of photocomposing control codes.

The output in Appendix XI was produced from the short Fortran program which is also listed in the Appendix together with the input data which the program used for the test.

The output in Appendix XII was produced by a program that constructs a numerical representation of a table of algebraic formulae, that is discussed elsewhere.[5]

IX. BUILT UP FORMULAE

With the exception of the example of Appendix XII, relatively little has been done to date by the authors on the setting of built up mathematical and chemical formulae. The construction of a convenient and intelligible linear representation of such formulae, that could be punched on a keyboard machine such as a Flexowriter, with a limited character set, is not trivial. The problem of linearizing formulae is really a special case of the more general problem of linearizing two dimensional topologies. A scheme is being developed by the authors, that gives names to objects, and then compounds these names in expressions in which connective symbols and operators are used to indicate topological association, scaling and alignment. Names are given to composite objects that are represented by these expressions, and these names are then used to form further expressions. Page composition provides a descriptive problem that is less serious, as the subdivision of pages into rectangles of material that can be set as units usually is simpler than the corresponding subdivision of formulae. A simple notation can be used to describe a page by an expression which uses names for items of text, and a nested algebraic notation for the distribution of these, in adjacent rectangles of different sizes.

X. VERBAL PROCESSING

Verbal processing by digital computer is of potential importance in many fields. Although mechanical translation and automatic abstracting have attracted considerable attention, the usefulness of the results is still being assessed. There are numerous processes of a more routine nature, however, that are certainly within

the scope of existing computer techniques. Updating verbal texts by reference to editorial commands of a relatively simple type has been discussed elsewhere.[6] More elaborate editing systems provide an interesting programming challenge. The production of indexes by the extraction of items from records of a reasonably standard format, and alphabetic sorting, is another problem of widespread interest. The definition of convenient languages for the specifications of such problems to general purpose programs that could be used for mechanized documentation requires further consideration.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues in the Cooperative Computing Laboraatory, Department of Libraries, Publication Department and Technology Press at MIT for the benefit of helpful discussion and their contribution to specific aspects of the work that is described here. Thanks are due to the Machine Composition Company of Boston for their cooperation in the processing of Photon tapes. This work has been supported in part by a grant RG 10430 of the U. S. Public Health Services, National Institutes of Health.

REFERENCES

1. M. P. BARNETT and R. P. FUTRELLE, Syntactic Analysis by Digital Computer, Comm. A. C. M. 5, 515, 1962.
2. M. P. BARNETT, K. L. KELLEY and M. J. BAILEY, Computer Generation of Photocomposing Control Tapes. Part I, The PC1 and PC2 Systems, Amer. Doc., 13, 58, 1962.
3. M. P. BARNETT, D. J. MOSS and D. A. LUCE, The Structure of the PC6 Photocomposing Program, Cooperative Computing Laboratory Technical Note No. 29, MIT, 1963.
4. M. P. BARNETT, D. J. MOSS and D. A. LUCE, Computer Generation of Photocomposing Control Tapes. Part II. The PC6 System, Cooperative Computing Laboratory Technical Note No. 28, MIT, 1963.
5. M. P. BARNETT, The Evaluation of Molecular Integrals by the Zeta Function Method, Chapter in Methods of Computational Physics, Vol. II, Academic Press, 1963.
6. M. P. BARNETT and K. L. KELLEY, Computer Editing of Verbal Texts. Part I. The ESI System, Amer. Doc., April, 1963.

APPENDIX IA

Flexowriter Input

```
LL36D1L 18CEN
Computer Controlled Printing Devices

L 12RLJEI4
```

A set of programs has been developed with the cooperation of the Machine Composition Company, and Photon Incorporated, that enable a digital computer to produce, as output, a punched paper tape to control the operation of the Photon photocomposing machine. This machine is used extensively in the printing of books and periodicals which contain verbal texts and mathematical, chemical and other symbolic material. The paper tape is punched with codes that determine the choice, size and spacing of the successive symbols that are to appear on the printed page. During any one continuous operation of the photocomposing machine, a total of 1440 different letters, digits and symbols can be selected in any sequence and in any two dimensional arrangement that is necessary. EL0

EL4 The computer can produce output to control the Photon machine in three ways. The first is by converting the numerical results of calculations, which the computer has effected, into Photon code. The second is by converting, to the Photon code, information that has been read into the computer, in some other code, on punched cards, paper tape, or other input media. The third is by reading into the computer information expressed in Photon code and punched on paper tape that has been produced by the computer in earlier operations of these three types. EL0

APPENDIX IB

Photocomposition

Computer Controlled Printing Devices

A set of programs has been developed with the cooperation of the Machine Composition Company, and Photon Incorporated, that enable a digital computer to produce, as output, a punched paper tape to control the operation of the Photon photocomposing machine. This machine is used extensively in the printing of books and periodicals which contain verbal texts and mathematical, chemical and other symbolic material. The paper tape is punched with codes that determine the choice, size and spacing of the successive symbols that are to appear on the printed page. During any one continuous operation of the photocomposing machine, a total of 1440 different letters, digits and symbols can be selected in any sequence and in any two dimensional arrangement that is necessary.

The computer can produce output to control the Photon machine in three ways. The first is by converting the numerical results of calculations, which the computer has effected, into Photon code. The second is by converting, to the Photon code, information that has been read into the computer, in some other code, on punched cards, paper tape, or other input media. The third is by reading into the computer information expressed in Photon code and punched on paper tape that has been produced by the computer in earlier operations of these three types.

The first of these methods enables the Photon to be used essentially as a powerful output device for a computer. Attention has been directed during the past quarter to the second of these methods using the computer to organize the information needed to control the Photon in the printing of verbal text, mathematical and chemical equations, and so forth from a less readable representation of such material prepared on a flexowriter. At present the flexowriter tape must be converted to punched cards in a trivial preliminary operation, and the computer output on punched cards converted to paper tape in another comparable operation. These stages will be bypassed later, using paper tape input-output attachments on the 709.

Completely verbal material can be typed on a flexowriter that produces a typescript similar to that of a conventional typewriter (fixed letter-width, single letter style, no justification) and a paper tape punched with a representation of this text. This can be converted by the computer to the tape that controls the Photon in an operation which prints the requisite text in any selected letter style and with justification if necessary. Comments may be interspersed in the text or appended to it that relate to changes of letter style or size, format control and so forth and these used by the computer to produce a tape which causes the Photon to print the text in the manner that the comments specify (the comments, of course, are not printed).

APPENDIX IIA

Flexowriter Input

```
[IN][DL2][LS10][LN500]I. Introduction[NP]This paper describes some of the
characteristics and applications of programs that have been developed
 recently in the author's laboratory, for the production of coded pa
per tapes to control the Photon photocomposing machine. Conventional
typesetting machines have been supplemented in the last two decades by
a variety of photocomposing machines that produce the original copy for
photolithographic reproduction by a photographic process. In this proc
ess, images of letters and characters are focused by an optical system
at appropriate positions of a roll of sensitized paper. A photocompo
sing machine contains a matrix of transparent characters in an opaque
background, with a mechanism for illuminating one selected character
at a time. Several types of photocomposing machines have been designed
and manufactured. The work that is reported here has used some Photon
machines that are installed in Boston, and which are equipped with paper
tape readers.[NP]The present photocomposing programs enable Photon
paper tapes to be punched from 709 output that is formed from input
that was read from Hollerith cards or Flexowriter tape, or which was
formed within the computer in binary coded decimal form by conversion
from an internal number representation. This makes it possible to phot
ocompose conventional computer results of a numerical nature, and to use
the computer to organize verbal and other material in routine ways, for
subsequent photocomposition. This organization may be the mechanical im
position of format requirements that would require elaborate manual type
setting, on material that is punched on a simple keyboard device, such as
Flexowriter, with a separate description of output format, or interspersed
parenthetic comments to specify format changes. The organization may
entail selection, extraction and sorting of items of information, and
more elaborate operations of verbal processing. The terms selection and e
xtraction are used with specialized meanings--the selection of records which sa
tisfy certain criteria (such as the occurrence of certain words) and
the extraction of specified portions of successive records, (for example
the extraction of author names form abstracts of journal articles).[EN]
```

APPENDIX IIb

Photocomposition

I. Introduction

This paper describes some of the characteristics and applications of programs that have been developed recently in the author's laboratory, for the production of coded paper tapes to control the Photon photocomposing machine. Conventional typesetting machines have been supplemented in the last two decades by a variety of photocomposing machines that produce the original copy for photolithographic reproduction by a photographic process. In this process, images of letters and characters are focused by an optical system at appropriate positions of a roll of sensitized paper. A photocomposing machine contains a matrix of transparent characters in an opaque background, with a mechanism for illuminating one selected character at a time. Several types of photocomposing machines have been designed and manufactured. The work that is reported here has used some Photon machines that are installed in Boston, and which are equipped with paper tape readers.

The present photocomposing programs enable Photon paper tapes to be punched from 709 output that is formed from input that was read from Hollerith cards or Flexowriter tape, or which was formed within the computer in binary coded decimal form by conversion from an internal number representation. This makes it possible to photocompose conventional computer results of a numerical nature, and to use the computer to organize verbal and other material in routine ways, for subsequent photocomposition. This organization may be the mechanical imposition of format requirements that would require elaborate manual typesetting, on material that is punched on a simple keyboard device, such as Flexowriter, with a separate description of output format, or interspersed parenthetic comments to specify format changes. The organization may entail selection, extraction and sorting of items of information, and more elaborate operations of verbal processing. The terms selection and extraction are used with specialized meanings--the selection of records which satisfy certain criteria (such as the occurrence of certain words) and the extraction of specified portions of successive records, (for example the extraction of author names form abstracts of journal articles).

APPENDIX III

Hollerith Input

```
*IN,LS10,DL2,JU,*$CHEMISTRY *NL,VL2*T$HE $D$EPARTMENT OF $C$HEMISTRY OFF
ERS A SINGLE UNDERGRADUATE $C$OURSE, SUFFICIENTLY FLEXIBLE IN ITS ELECTI
VES SO THAT IT PROVIDES EXCELLENT PREPARATION FOR CAREERS IN MANY DIFFER
ENT AREAS OF CHEMISTRY, AND GRADUATE PROGRAMS FOR THREE ADVANCED DEGREES
. $T$HERE ARE EXCELLENT OPPORTUNITIES FOR STUDY AND RESEARCH IN PHYSICA
L, ORGANIC, NUCLEAR, AND ANALYTICAL CHEMISTRY. $I$N ADDITION, THE $D$EPA
RTMENT IS RESPONSIBLE FOR UNDERGRADUATE AND GRADUATE INSTRUCTION IN CHEM
ISTRY FOR STUDENTS IN MANY OTHER $I$NSTITUTE $C$OURSES. *VL5,EN*
```

Photocomposition

CHEMISTRY

The Department of Chemistry offers a single undergraduate Course, sufficiently flexible in its electives so that it provides excellent preparation for careers in many different areas of chemistry, and graduate programs for three advanced degrees. There are excellent opportunities for study and research in physical, organic, nuclear, and analytical chemistry. In addition, the Department is responsible for undergraduate and graduate instruction in chemistry for students in many other Institute Courses.

APPENDIX IV

Flexowriter Input

```
[indn77d12ls24st1,,36cnxs1]
EXCERPT FROM ALICE IN WONDERLAND
[nllsl8]
December 6, 1961
[sp4st2,10,36st3,11,36st4,12,36st5,13,36st6,14,36st7,15,36
st8,16,36st9,17,36st10,18,36ls14dllxs2rl]
[scl9scl9]Fury said to
[xs3]a mouse, That
[nlxs6]he met
[xs7]in the
[xs8]house,
[xs9][scl9]Let us
[xs7]both go
[xs6ls12]to law:
[xs5d19]I[dll] will
[xs3]prosecute
[xs2d19]you.
[nlxs3dll] Come, I'll
[nlxs4]take no
[nlxs5]denial:
[nlxs7]We must
[nlxs8ls11]have a
[nlxs9]trial[sc47]
[nlxs10] For
[xs7]really
[xs6]this
[nl] morning
[nlxs6ls10] I've
[xs6]nothing
[xs5]to do.'
[xs4]Said the
[xs2]mouse to
[xsl] the cur,
[nlxs3][scl9]Such a
[nlxs4ls9]trial,
[xs3]dear sir,
[xs2] With no
[xs2]jury or
[xsllsS]judge,
[nlxs2]would be
[xs2] wasting
[xs2] our breath.'
[xs3ls7][scl9]I'll be
[xs2] judge,
[xs2] I'll be
[xs2]jury,'
[xsl] Said
[xsllsS] cunning
[xsl] old Fury:
[xs2][scl9]I'll try
[xs2]the whole
[nlxs3ls5] cause,
[nlxs4]and
[xs3] condemn
[xs3]you
[xs3]to
[xs3] death.'''
[en]
```

Photocomposition

"Fury said to
a mouse, That
he met
in the
house,
'Let us
both go
to law:
I will
prosecute
you.
Come, I'll
take no
denial:
We must
have a
trial;
For
really
this
morning
I've
nothing
to do.'
Said the
mouse to
the cur,
'Such a
trial,
dear sir,'
With no
jury or
judge,
would be
wasting
our breath.
'I'll be
judge,
I'll be
jury,'
Said
cunning
old Fury:
'I'll try
the whole
cause,
and
condemn
you
to
death.'"

APPENDIX V

Photocomposition

u	v	w	m	n	a coefficient
0	0	0	0	0	1/1
0	0	1	1	0	1/1
0	0	2	0	0	1/3
			2	0	2/3
0	0	3	1	0	3/5
			3	0	2/5
0	0	4	0	0	1/5
			2	0	4/7
			4	0	8/35
0	1	0	1	1	-1/1
			4	0	8/35
0	1	1	2	1	-1/3
			3	0	-2/15
			3	2	-1/45
			5	0	1/21
			5	2	1/315
			5	4	1/7,560
			7	1	4/1,001
			7	3	2/45,045
0	4	2	0	0	1/35
			2	2	-1/63
			4	0	-3/55
			4	2	-1/231
			4	4	1/9,240
			6	0	2/77
			6	2	4/3,465
			6	4	1/41,580
0	4	3	1	0	1/35
			3	0	-4/165
			3	2	-1/165
			5	0	-5/273
			5	2	-1/1,365
			5	4	1/32,760
			7	0	2/143
			7	2	4/9,009
			7	4	1/180,180
0	4	4	0	0	1/105
			2	0	2/231
			2	2	-1/231
			4	0	-111/5,005
			4	2	-1/455
			4	4	1/40,040
			6	0	-4/1,155
			6	4	1/103,950
			8	0	16/2,145
			8	2	8/45,045
			8	4	1/675,675
1	0	0	1	1	-1/1
			3	1	-4/45
			5	1	-8/315

APPENDIX VI

BCD Input

```
-     II
0     AT THE ANNUAL CASC (COUNCIL FOR THE ADVANCEMENT OF SMALL
      COLLEGES) CONFERENCE IN AUGUST, THE THEME WAS FACILITIES.  A
      REPORT ON THE GASP PROJECT WAS MADE AT THAT TIME.  THE IDEA
      THAT COMPUTER SIMULATION TECHNIQUES COULD BE USED IN STUDYING
      FACILITIES WAS INTRODUCED AND OUR SPACE UTILIZATION STUDY
      REPORT (ISSUED 16 JULY WITH THE SECOND GASP PROGRESS REPORT)
      WAS DISCUSSED.
-     III
0     A LABORATORY COURSE IN OPERATIONS RESEARCH METHODS IS GIVEN
      AT M.I.T.  IN WHICH PROBLEMS WITHIN THE INSTITUTE AMENABLE
      TO SOLUTION BY SUCH TECHNIQUES ARE CONSIDERED.  THIS TERM A
      PART OF THAT CLASS IS STUDYING SOME ASPECTS OF THE SCHEDULING
      PROBLEM UNDER THE DIRECTION OF DR. H. P. GALLIHER.  THERE
      HAS BEEN, AND WILL CONTINUE TO BE, MUCH CONTACT BETWEEN
      THIS GROUP AND THE GASP PROJECT.
```

Photocomposition

II

AT THE ANNUAL CASC (COUNCIL FOR THE ADVANCEMENT OF SMALL COLLEGES) CONFERENCE IN AUGUST, THE THEME WAS FACILITIES. A REPORT ON THE GASP PROJECT WAS MADE AT THAT TIME. THE IDEA THAT COMPUTER SIMULATION TECHNIQUES COULD BE USED IN STUDYING FACILITIES WAS INTRODUCED AND OUR SPACE UTILIZATION STUDY REPORT (ISSUED 16 JULY WITH THE SECOND GASP PROGRESS REPORT) WAS DISCUSSED.

III

A LABORATORY COURSE IN OPERATIONS RESEARCH METHODS IS GIVEN AT M.I.T. IN WHICH PROBLEMS WITHIN THE INSTITUTE AMENABLE TO SOLUTION BY SUCH TECHNIQUES ARE CONSIDERED. THIS TERM A PART OF THAT CLASS IS STUDYING SOME ASPECTS OF THE SCHEDULING PROBLEM UNDER THE DIRECTION OF DR. H. P. GALLIHER. THERE HAS BEEN, AND WILL CONTINUE TO BE, MUCH CONTACT BETWEEN THIS GROUP AND THE GASP PROJECT.

APPENDIX VII

Photocomposition

11	Astrojet	Dallas, Texas	DAL
12	Boeing	Danville, Ill.	DNV
13	Convair	Danville, Va.	DAN
14	Douglas	Dauphin, Manitoba	YDN
15	Electra	Dawson City, Y.T.	YDA
16	Fairchild	Dawson Creek, Br.Col.	YDQ
17	Golden Falcon	Dayton, Ohio	DAY
18	Helio	Daytona Beach, Fla.	DAB
19	Jetaway Vacations	Decatur, Ala.	DCU
20	Jet-Powered	Decatur, Ill.	DEC
21	Lockheed	Deering, Alaska	DRG
22	Mainliner	Del Monte, Calif.	MRY
23	Non-Stop	Delta, Colo.	MTJ
24	On-Time	Denison, Texas	SWI
25	Prop-Jet	Denver, Colo.	DEN
26	Qantas	De Ridder, Louisiana	DRI
27	Red Carpet	Des Moines, Iowa	DSM
28	Sunliner	Destin, Fla.	VPS
29	Tourist	Detroit, Mich.	DTW
30	Vickers	Devils Lake, N.D.	DVL

11	*Astrojet*	*Dallas, Texas*	*DAL*
12	*Boeing*	*Danville, Ill.*	*DNV*
13	*Convair*	*Danville, Va.*	*DAN*
14	*Douglas*	*Dauphin, Manitoba*	*YDN*
15	*Electra*	*Dawson City, Y.T.*	*YDA*
16	*Fairchild*	*Dawson Creek, Br.Col.*	*YDQ*
17	*Golden Falcon*	*Dayton, Ohio*	*DAY*
18	*Helio*	*Daytona Beach, Fla.*	*DAB*
19	*Jetaway Vacations*	*Decatur, Ala.*	*DCU*
20	*Jet-Powered*	*Decatur, Ill.*	*DEC*
21	*Lockheed*	*Deering, Alaska*	*DRG*
22	*Mainliner*	*Del Monte, Calif.*	*MRY*
23	*Non-Stop*	*Delta, Colo.*	*MTJ*
24	*On-Time*	*Denison, Texas*	*SWI*
25	*Prop-Jet*	*Denver, Colo.*	*DEN*
26	*Qantas*	*De Ridder, Louisiana*	*DRI*
27	*Red Carpet*	*Des Moines, Iowa*	*DSM*
28	*Sunliner*	*Destin, Fla.*	*VPS*
29	*Tourist*	*Detroit, Mich.*	*DTW*
30	*Vickers*	*Devils Lake, N.D.*	*DVL*

11	**Astrojet**	**Dallas, Texas**	**DAL**
12	**Boeing**	**Danville, Ill.**	**DNV**
13	**Convair**	**Danville, Va.**	**DAN**
14	**Douglas**	**Dauphin, Manitoba**	**YDN**
15	**Electra**	**Dawson City, Y.T.**	**YDA**
16	**Fairchild**	**Dawson Creek, Br.Col.**	**YDQ**
17	**Golden Falcon**	**Dayton, Ohio**	**DAY**
18	**Helio**	**Daytona Beach, Fla.**	**DAB**
19	**Jetaway Vacations**	**Decatur, Ala.**	**DCU**
20	**Jet-Powered**	**Decatur, Ill.**	**DEC**
21	**Lockheed**	**Deering, Alaska**	**DRG**
22	**Mainliner**	**Del Monte, Calif.**	**MRY**
23	**Non-Stop**	**Delta, Colo.**	**MTJ**
24	**On-Time**	**Denison, Texas**	**SWI**
25	**Prop-Jet**	**Denver, Colo.**	**DEN**
26	**Qantas**	**De Ridder, Louisiana**	**DRI**
27	**Red Carpet**	**Des Moines, Iowa**	**DSM**
28	**Sunliner**	**Destin, Fla.**	**VPS**
29	**Tourist**	**Detroit, Mich.**	**DTW**
30	**Vickers**	**Devils Lake, N.D.**	**DVL**

APPENDIX VIII

Flexowriter Input

616.21
Chu-hua erh pi yen hou k/o tsa chih (Zhonghua erbiyanhouke zazhi)
[Chinese Journal of Otorhinolaryngology]
Peking, People's Medical Publishers, [Aug. 1953], varies,bimonthly

DNLM	v.2, 4, 1954 v.8, 1-3, 1960	v.3-7, 1955-59	
MCM		v.3-7, 1955-59	v.8, 1-3, 1960
MH-HY	v.2, 4, 1954	v.3-7, 1955-59	
GbBM	v.4, 4, 1956		

615.84
Chung-hua fang she hsuΣeh tsa chih (Zhonghua fangshexue zazhi)
[Chinese Journal of Radiology]
Peking, People's Health Press, [Sept. 1953], varies,bimonthly

DNLM	v.2, 2,4, 1954 v.8, 1,2, 1960	v.3-7, 1955-59	
MCM		v.3, 1-3, 1955	v.4-7, 1956-59
	v.8, 1,2, 1960		
MH-HY	v.2, 4, 1954 v.5, 1957	v.4, 1956	
GbDSIR	v.7, 4-6, 1959		
HkURI	v.4, 1956	v.5, 1-3, 1957	

618
Chung-hua fu chΛan k/o tsa chih (Zhonghua fuchanke zazhi)
[Chinese Journal of Obstetrics and Gynecology]
Peking, People's Medical Publishers, [Apr. 1953], varies,bimonthly

DNLM	v.2, 3,4, 1954 v.6, 1-5, 1958 v.8, 1,2, 1960	v.3-5, 1955-57 v.7, 1959	
MCM		v.3, 2-4, 1955	v.4, 1956
	v.5, 1957	v.6, 1-5, 1958	
	v.7, 1959	v.8, 1,2, 1960	
MH-HY	v.2, 4, 1954	v.4, 1956	
	v.5, 1957	v.6, 1-5, 1958	
	v.7, 1959		
JpNDL	v.6, 1958		

APPENDIX VIII

Photocomposition

004
Chung-hua erh pi yen hou k'o tsa chih (Zhonghua erbiyanhouke zazhi)
[Chinese Journal of Otorhinolaryngology]
Peking, People's Medical Publishers, [Aug. 1953], varies,bimonthly

DNLM	v.2, 4, 1954	v.3-7, 1955-59
	v.8, 1-3, 1960	
MCM	v.3-7, 1955-59	v.8, 1-3, 1960
MH-HY	v.2, 4, 1954	v.3-7, 1955-59
GbBM	v.4, 4, 1956	

005
Chung-hua fang she hsueh tsa chih (Zhonghua fangshexue zazhi)
[Chinese Journal of Radiology]
Peking, People's Health Press, [Sept. 1953], varies,bimonthly

DNLM	v.2, 2,4, 1954	v.3-7, 1955-59
	v.8, 1,2, 1960	
MCM	v.3, 1-3, 1955	v.4-7, 1956-59
	v.8, 1,2, 1960	
MH-HY	v.2, 4, 1954	v.4, 1956
	v.5, 1957	
GbDSIR	v.7, 4-6, 1959	
HkURI	v.4, 1956	v.5, 1-3, 1957

006
Chung-hua fu ch'an k'o tsa chih (Zhonghua fuchanke zazhi)
[Chinese Journal of Obstetrics and Gynecology]
Peking, People's Medical Publishers, [Apr. 1953], varies,bimonthly

DNLM	v.2, 3,4, 1954	v.3-5, 1955-57
	v.6, 1-5, 1958	v.7, 1959
	v.8, 1,2, 1960	
MCM	v.3, 2-4, 1955	v.4, 1956
	v.5, 1957	v.6, 1-5, 1958
	v.7, 1959	v.8, 1,2, 1960
MH-HY	v.2, 4, 1954	v.4, 1956
	v.5, 1957	v.6, 1-5, 1958
	v.7, 1959	
JpNDL	v.6, 1958	

007
Chung-hua i hsueh tsa chih (Zhonghua yixue zazhi)
[National Medical Journal of China]
Peking, People's Health Press, [1914], monthly

CSt-H	v.39, 6-12, 1953	v.40-44, 1954-58
DLC	v.37, 6-8, 1951	v.38, 7-12, 1952
	v.39-44, 1953-58	
DNLM	v.35, 1-8,10-12, 1949	v.37, 8,11,12, 1951
	v.38,1-3,7,8,10,12, 1952	v.39, 2,4-6,8-12, 1953
	v.44, 1958	v.46, 1, 1960
MCM	v.35, 1949	v.36, 1950

APPENDIX IXa

Flexowriter Input

[1,4025]DAVID MOSS MITCCL 1962[4]GEOLOGICAL LISTING[19,4720,5] 280[16]BIBLIOGRAPHY OF NORTH AMERICAN GEOLOGY, 1959[20,5]Summerson, Charles Henry.[29][30]1. Evidence of weathering at the Silurian-Devonian contact in central Ohio: Jour. Sed. Petrology, v. 29, no. 3, p. 425-429, illus, Sept. 1959.[17]2. Pre-glacial residual soil in central Ohio: Jour. Sed. Petrology, v. 29, no. 3, p. 430-435, illus., Sept. 1959.[18]Sun, Ming-Shan.[28][30]1. (and Weege, Ran dall J.). Native selenium from Grants, New Mexico: Am. Mineralogist, v. 44, nos. 11-12, p. 1309-1311, illus., Nov.-Dec. 1959.[17]2. Deter mination of selenium by X-ray spectroscopic method [23]abs.[24], [21]in[22] N. Mex. Geol. Soc., Guidebook, 10th Field Conf., Oct. 1959, p. 157, 1959.[18]Sun, Shiou Chuan. [21]See[2] Spokes, E. M. [31]Sund, J. Olaf.[28][30]Origin of the New Brunswick gypsum deposits: Canadian Min. Metall. Bull ., no. 571, p. 707-712, illus., Nov. 1959: Canadian Inst. Mining and Metallurgy Trans., v. 62, p. 395-400, illus., 1959.[18]Sundelius, Harold Wesley.[28] [30]Occurrence and origin of the Peg Claims spodumene pegmatites, Knox County, Maine [23]abs.[24]: Dissert. Abs., v. 20, no. 2, p. 642, Aug. 1959.[18]Sundius, Nils. [21]See[22] Vogt, T.[31] Susuki, Takeo. [21]See[22] Crowell, J. C., 1; Valentine, J. W., 2.[31]Suter, Max.[28][30](and others). Preliminary report on ground-water resources of the Chicago region, Illinois: Ill. State Water Survey Cooperative Ground-Water Rept. 1, 89 p., illus. incl. geol. maps, 1959, summary, Rept. 1-S, 18 p., illus., 1959.[18]

APPENDIX IXB

Photocomposition

280 BIBLIOGRAPHY OF NORTH AMERICAN GEOLOGY, 1959

Summerson, Charles Henry.

1. Evidence of weathering at the Silurian-Devonian contact in central Ohio: Jour. Sed. Petrology, v. 29, no. 3, p. 425-429, illus, Sept. 1959.
2. Pre-glacial residual soil in central Ohio: Jour. Sed. Petrology, v. 29, no. 3, p. 430-435, illus., Sept. 1959.

Sun, Ming-Shan.

1. (and Weege, Randall J.). Native selenium from Grants, New Mexico: Am. Mineralogist, v. 44, nos. 11-12, p. 1309-1311, illus., Nov.-Dec. 1959.
2. Determination of selenium by X-ray spectroscopic method [abs.], *in* N. Mex. Geol. Soc., Guidebook, 10th Field Conf., Oct. 1959, p. 157, 1959.

Sun, Shiou Chuan. *See* Spokes, E. M.

Sund, J. Olaf.

Origin of the New Brunswick gypsum deposits: Canadian Min. Metall. Bull., no. 571, p. 707-712, illus., Nov. 1959, Canadian Inst. Mining and Metallurgy Trans., v. 62, p. 395-400, illus., 1959.

Sundelius, Harold Wesley.

Occurrence and origin of the Peg Claims spodumene pegmatites, Knox County, Maine [abs.]: Dissert. Abs., v. 20, no. 2, p. 642, Aug. 1959.

Sundius, Nils. *See* Vogt, T.

Susuki, Takeo. *See* Crowell, J. C., 1, Valentine, J. W., 2.

Suter, Max.

(and others). Preliminary report on ground-water resources of the Chicago region, Illinois: Ill. State Water Survey Cooperative Ground-Water Rept. 1, 89 p., illus. incl. geol. maps, 1959, summary, Rept. 1-S, 18 p., illus., 1959.

Sutherland, Patrick Kennedy.

1. (and Amsden, Thomas William). A re-illustration of the trilobite *Lonchodomas mcgeheei* Decker from the Bromide formation (Ordovician) of southern Oklahoma: Okla. Geology Notes, v. 19, no. 10, p. 212-219, illus. incl. geol. sketch map, Oct. 1959.
2. (and Land, Cooper B., Jr.). Mississippian limestone boulder conglomerates in the southernmost Sangre de Cristo ountains, New Mexico [abs.]: Geol. Soc. America Bull., v. 70, no. 12, pt. 2, p. 1683, Dec. 1959.

Sutherland, Pauline.

New England's new desert [Maine]: Desert Mag., v. 22, no. 2, p. 10-11, illus., Feb. 1959.

Sutterlin, Peter George.

A stratigraphic analysis of the Winterburn and Wabamun groups in southern Alberta, *in* Am. Assoc. Petroleum Geologists Rocky Mtn. Sec., Geological record, Feb. 1959, p. 17-23, illus. [1959].

Sutton, George H. *See* Drake, C. L., Talwani, M., 2.

Sutton, Robert George.

1. Use of flute casts in stratigraphic correlation [N.Y.]: Am. Assoc. Petroleum Geologists Bull., v. 43, no. 1, p. 230-237, illus., Jan. 1959.
2. Structural geology of the Dryden and Harford quadrangles, New York. 15 p., illus., Albany, Univ. State N. Y., July 1959.

Swain, Frederick Morrill, Jr. *See also* Dobbins, D. A.: Palacas, J. G.

1. (and Blumentals, A., and Millers, R.). Stratigraphic distribution of amino acids in peats from Cedar Creek Bog, Minnesota, and Dismal Swamp, Virginia: Limnology and Oceanography, v. 4, no. 2, p. 119-127, illus., Apr. 1959.
2. Amino acid distribution in lake deposits [abs.]: Minn. Univ. Center Continuation Study Inst. Lake Superior Geology, 5th Ann. Mtg., Apr. 13-14, 1959, p. 13(‡) [1959]

APPENDIX Xa

Hollerith Input

```
A.E.G. PROGRESS                                                    41611      7
    (ALLGEMEINE ELEKTRICITATSGESELLSCHAFT, BERLIN)                0041621      7
NO.1,APRIL 1925-1934          ENG G      VAIL                     004163B      7
1956,NO.1+                                                         41642      7
A.E.I. ENGINEERING                                                 42011      7
     (ASSOCIATED ELECTRICAL INDUSTRIES, LTD.)                      0042021      7
V.1,1961+                     ENG G      VAIL                     004203B      7
A.I.B.S. BULLETIN                                                  43211      2
     (AMERICAN INSTITUTE OF BIOLOGICAL SCIENCES)                   0043221      2
V.1,1951+                     SCIENCE    BIOL P 570.51 A51        004323B      2
A.I.CH.E. JOURNAL                                                  44811      2
     (AMERICAN INSTITUTE OF CHEMICAL ENGINEERS)                    0044821      2
V.1,NO.1,MAR 1955+            SCIENCE    660.51 A11               004483B      2
A.I.P. DOCUMENTATION NEWSLETTER                                    45611      3
     (AMERICAN INSTITUTE OF PHYSICS)                               0 45621      3
V.1,NO.1,JULY 1959+           *HAYDEN    P 530.51 A11             004563B      3
A.O.P.A. PILOT                                                     51211      1
     (AIRCRAFT OWNERS AND PILOTS ASSOCIATION)                      0051221      1
V.2,NO.8,AUG 1959+            AERO                                0 5123B      1
A.P.C.A. ABSTRACTS                                                 52811      7
     (AIR POLLUTION CONTROL ASSOCIATION)                           0052821      7
V.2,NO.4,SEPT,                ENG G                               0 5283B      7
NO.6,NOV 1956+                                                     52842      7
A.S.E.A.                                                           57611
     (ALLMANNA SVENSKA ELEKTRISKA AKTIEBOLAGET)                    0057621
     SEE ASEA                                                      57631
A.S.H.R.A.E. JOURNAL                                               59211      7
     (AMERICAN SOCIETY OF HEATING, REFRIGERATING                   0059221      7
     AND AIR-CONDITIONING ENGINEERS)                               0 59231      7
     EARLIER SEE REFRIGERATING ENGINEERING                         0059241      7
V.1,NO.3,MAR 1959-NO.12,      HAYDEN     P 628.8051 A11           005925B      3
DEC 1959                                                           59262      3
V.1,NO.3,MAR 1959+            ENG G                               0 5927K      7
A.T.E. JOURNAL                                                     60811      7
     (AUTOMATIC TELEPHONE AND ELECTRIC COMPANY)                    0060821      7
V.8,1951+                     ENG G      VAIL                     006083B      7
```

APPENDIX Xb

Photocomposition

TITLE	HOLDINGS	LIBRARY	CALL NUMBER
A.E.G. Progress (Allgemeine Elektricitatsgesellschaft, Berlin)	no.1,April 1925-1934 1956,no.1+	ENG G	Vail
A.E.I. Engineering (Associated Electrical Industries, Ltd.)	v.1,1961+	ENG G	Vail
A.I.B.S. Bulletin (American Institute of Biological Sciences)	v.1,1951+	SCIENCE	Biol P 570.51 A51
A.I.Ch.E. Journal (American Institute of Chemical Engineers)	v.1,no.1,Mar 1955+	SCIENCE	660.51 A11
A.I.P. Documentation Newsletter (American Institute of Physics)	v.1,no.1,July 1959+	*HAYDEN	P 530.51 A11
A.O.P.A. Pilot (Aircraft Owners and Pilots Association)	v.2,no.8,Aug 1959+	AERO	
A.P.C.A. Abstracts (Air Pollution Control Association)	v.2,n .4,Sept, no.6,Nov 1956+	ENG G	
A.S.E.A. (Allmanna Svenska Elektriska Aktiebolaget) See Asea			
A.S.H.R.A.E. Journal (American Society of Heating, Refrigerating and Air-Conditioning Engineers) Earlier See Refrigerating Engineering	v.1,no.3,Mar 1959-no.12, Dec 1959	HAYDEN	P 628.8051 A11
	v.1,no.3,Mar 1959+	ENG G	
A.T.E. Journal (Automatic Telephone and Electric Company)	v.8,1951+	ENG G	Vail
A.W.A. Technical Review (Amalgamated Wireless, Ltd.)	v.3,1937+	ENG G	Vail
Abhandlungen in Addition to the following Title See Abhandlungen Under Names of Societies and Institutions			
Abhandlungen aus der Sowjetischen Physik	v.1,1951+	SCIENCE	
Abo (Finland). Akademi. Acta. Mathematica et Physica	v.4,1927, v.12,1940+	SCIENCE	510.6945 A15 Aa
Abstracts Journal of Metallurgy	1957,no.1+	SCIENCE	Lindgren 016.669 A16
Abstracts of English Studies	v.1,1958+	GEN + HUM	016.8 A16

APPENDIX XIa

Source Program and Data

```
*       XEQ
*       LIST
C       MAIN PROGRAM
        DIMENSION FORM(24),FORMA(12)
        READ INPUT TAPE 4,9,(FORM(K),K=1,24),(FORMA(K),K=1,12)
        COMPOSE FORM
    5   READ INPUT TAPE 4, 2, N1, N2
        IF (N1) 6, 7, 7
    7   N3 = N1 + N2
        COMPOSE 3, N1, N2, N3
        GO TO 5
    2   FORMAT (2I5)
    3   FORMAT (2KRJ I3, 3H +  I3, 5KXS3RL 3H =  I4, 3KXS2)
    6   COMPOSE FORMA
        CALL EXIT
    9   FORMAT (12A6)
        END
*       DATA
(21KINDL2LS18ST1,,36XS1CN 50H$SIMPLE ARITHMETIC TO DEMONSTRATE PHOTOCOMP
OSITION 7KXS1AU15 15H$D$ECEMBER 1961 29KSP3DL1LS12ST2,,18ST3,18,36XS2)
(5KXS1CN 4H$END 4KNLEN)
    1    1
    1    2
   27   54
   57   82
    7    2
    0    0
  100 1000
   26   14
   -1
```

APPENDIX XIb

Photocomposition

SIMPLE ARITHMETIC TO DEMONSTRATE PHOTOCOMPOSITION

December 1961

1 + 1 = 2

1 + 2 = 3

27 + 54 = 81

57 + 82 = 139

7 + 2 = 9

0 + 0 = 0

100 + 1000 = 1100

26 + 14 = 40

END

APPENDIX XII

Photocomposition

TABLE OF $e^{k,j,i}_{u+v,w}$ COEFFICIENTS

u+v	w	j	i	$e^{k,j,i}_{u+v,w}$
1	2	1	3	$2(k+2)(k+3)/(2k+1)(2k+3)(2k+5)$
2	1	0	-3	$-2k(k-1)(k-2)/(2k-1)(2k+1)(2k-3)$
2	1	0	-1	$2k(k^2-3)/(2k+1)(2k-3)(2k+3)$
2	1	0	1	$2(k^2+2k-2)(k+1)/(2k+1)(2k-1)(2k+5)$
2	1	0	3	$-2(k+1)(k+2)(k+3)/(2k+1)(2k+3)(2k+5)$
2	1	2	-3	$2(k-2)/(2k-1)(2k+1)(2k-3)$
2	1	2	-1	$-2(k-3)/(2k+1)(2k-3)(2k+3)$
2	1	2	1	$-2(k+4)/(2k+1)(2k-1)(2k+5)$
2	1	2	3	$2(k+3)/(2k+1)(2k+3)(2k+5)$
3	0	1	-3	$2(k-1)(k-2)/(2k-1)(2k+1)(2k-3)$
3	0	1	-1	$-2(3k^2-k-6)/(2k+1)(2k-3)(2k+3)$
3	0	1	1	$2(3k^2+7k-2)/(2k+1)(2k-1)(2k+5)$
3	0	1	3	$-2(k+2)(k+3)/(2k+1)(2k+3)(2k+5)$
3	0	3	-3	$-2/(2k-1)(2k+1)(2k-3)$
3	0	3	-1	$6/(2k+1)(2k-3)(2k+3)$
3	0	3	1	$-6/(2k+1)(2k-1)(2k+5)$
3	0	3	3	$2/(2k+1)(2k+3)(2k+5)$
0	4	0	-4	$2k(k-1)(k-2)(k-3)/(2k+1)(2k-1)(2k-3)(2k-5)$
0	4	0	-2	$4k(2k^2-2k-7)(k-1)/(2k-1)(2k+1)(2k+3)(2k-5)$
0	4	0	0	$6(2k^4+4k^3-6k^2-8k+3)/(2k-1)(2k-3)(2k+3)(2k+5)$
0	4	0	2	$4(2k^2+6k-3)(k+1)(k+2)/(2k+1)(2k-1)(2k+3)(2k+7)$
0	4	0	4	$2(k+1)(k+2)(k+4)(k+3)/(2k+1)(2k+3)(2k+5)(2k+7)$
1	3	1	-4	$-2(k-1)(k-2)(k-3)/(2k+1)(2k-1)(2k-3)(2k-5)$

ON THE SOLUTION OF AN INFORMATION RETRIEVAL PROBLEM

Burnett H. Sams
Data Systems Center
Radio Corporation of America
Bethesda 14, Maryland

The problem was to formulate a system which would be capable of digesting an input stream of documents in such a manner as to be able to regurgitate selected information in response to interrogations by a number of research analysts. A number of comments on the problem are in order. Portions of such systems are computers and computer programs serving as information processors. The documents are assumed to contain formatted texts on a sufficiently restricted subject matter to permit mechanical recognition and analysis of what for the moment will be loosely called the informational content of the document. That the system is intended to have a number of users is significant and implies that the person who puts information into the system and the person who takes it out are not always the same. Either there are very tight conventions governing the storage and retrieval of information, or there is a mechanism for associating classes of input and output descriptions. As the number of users or the scope of subject matter increases, it becomes increasingly necessary to provide leeway on taking in information and to provide alternative routes for getting at information stored within the system. In larger systems it may become necessary to accommodate man-machine dialogue enroute to desired information.

This paper is concerned with the design and programming of information retrieval systems. Typical units for measuring the capacity of random-access storage and the size of computer programs are respectively 10^8 characters and 10^5 instructions. The computer for this application is taken to be a general-purpose, stored-program, digital computer having a high-speed, random-access memory for program storage, working storage, and data storage of $2^{15} - 2^{17}$ words. The internal operations are geared to manipulate strings of bits. The applications will generally require serial magnetic tapes in addition to random-access memories. The computer has programable input and output units capable of operating in a simultaneous manner. This simultaneous processing capability is coordinated with internal processing by a program interrupt subsystem which automatically saves the machine registers on an interruption.[8]

The system discussed below is not an actual development but is an extension of developments leading to frontiers in information retrieval, programming, and computer technology. The ACSI-MATIC Program under contract with the Department of the Army, Office of the Assistant Chief of Staff for Intelligence is developing a major information system to support certain headquarters operations of the U. S. Army.[4, 6, 7, 9]

RETRIEVAL SYSTEMS

A distinction is made between a document retrieval system, an information retrieval system, and a collating information retrieval sys-

tem. In a document retrieval system, documents are stored as they are received, and the object is to retrieve documents having something in common. In an information retrieval system, documents are transformed and stored so as to render their "informational content" accessible to programs whose objective is to retrieve extracts from documents having something in common. In a collating information retrieval system, documents are not retained in the principal data store; instead, composites are formed of information extracted from many documents. Such a system has a potential for research on inference formation and input validation (Fig. 1).

System	Storage	Retrieval
Document	Documents	Documents
Information	Documents	Extracts
Collating	Composites	Extracts

Figure 1. Document retrieval, information retrieval and collating information retrieval systems are differentiated by their informational units of storage and retrieval.

In document retrieval systems generally, each document is indexed by a set of key terms. Likewise, each composite is indexed by a set of key terms. For example, if one wanted to know the current status of a particular hurricane, the document retrieval system would produce a few dozen weather reports each having something to say about the hurricane. The information retrieval system would produce extracts from the reports dealing with the hurricane. The collating system would have previously organized the data in the weather reports and would produce only the current status of the hurricane. If one asked about a tornado that was also discussed in those same weather reports, the document system would simply present the reports a second time. The information retrieval system would produce those extracts dealing with the tornado, and the collating system would limit itself to the status of the tornado.[3]

INDEXING

Each system requires some kind of indexing scheme that can locate records (documents, extracts, or composites) within a short period of time. The simplest kind of indexing associates an index term with a set of records each of which contains the index term. One may wish a more extensive association. For example, if one wants to get information about the consumption of electricity in New Jersey, one would not want to ignore a document concerning the electric consumption in Jersey City. The interrogator knows Jersey City is in New Jersey. He always wants related information about Jersey City or Newark or Trenton when he wants information about New Jersey. The index term "New Jersey," therefore, should subsume the index terms "Jersey City," "Newark," etc. This particular subsuming is tree-structured and one can devise various techniques for reflecting the desired relationships (Fig. 2). Sometimes a tree relationship of sub-

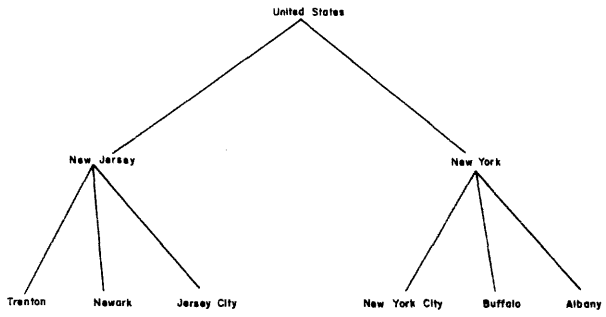


Figure 2. A section of a tree illustrating hierarchical relations among political units.

suming is not adequate. Suppose, in addition to having New Jersey subsume Jersey City, one also wants Greater New York to subsume Jersey City; Jersey City has now become a lattice point in the structure (Fig. 3). Process-

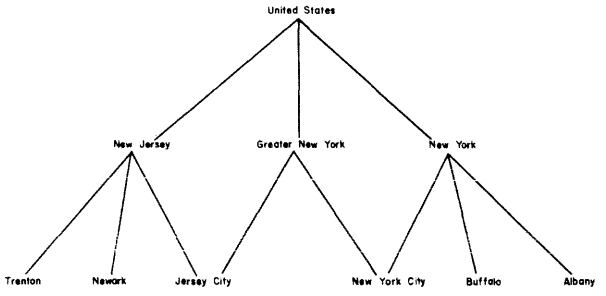


Figure 3. A section of a graph illustrating a non-tree-structured relation.

ing over an encoding of a lattice or a general linear graph is far more difficult and time-consuming than processing over an encoding of a tree; thus it is inefficient to view all points as graph points for the sake of generality. Each problem may have its own particular kinds of index term associations which may affect efficiency, and it is likely that most problems will have more than one kind. It is concluded, therefore, that information retrieval systems must have the ability to identify index terms as belonging to a class of index terms which permits a set of subsuming relations between elements in the class. One might also permit a set of subsuming relations to be defined among the classes themselves.

As soon as one permits classes of index terms, there is simultaneously a need for means of identifying the class to which a given term belongs and a need to identify terms within a class to see if they are legitimate and to find subsumed index terms. When the number of terms in a class is finite, a simple glossary will suffice. Each entry in the glossary could indicate which of the remaining entries in the glossary are subsumed index terms. When frequent additions to the glossary are anticipated, any scheme should be avoided that depends on the sequence of items in the glossary for these subsumed term indicators. If, however, the number of items in the class is infinite, one needs a recognition procedure. For example, suppose a system indexes all transactions of $1000 or more by indexing each document under the exact amount of the transaction. Suppose further that for standard requests one subsumes under a given amount all other amounts within a half-dollar of the given amount. Then—for an interrogation using $1065.00 as an index term, all amounts greater than $1064.49 and less than $1065.50 are subsumed.

Although the number of possible index terms in a system may be infinite, the number of documents or composities indexed and the number of index terms which actually reference documents are finite but unbounded. Each information retrieval system needs a list of active index terms with their coupling to the records they index. Schemes based upon searching the entire record store for each interrogation are excluded from these considerations of a large data store. The combination of glossaries and an index list is called a thesaurus. The index list is distinct from the glossaries, and some classes having an infinite number of index terms may not require glossaries. By the very fact that index terms may be active or inactive at a particular time, it is necessary to have means for updating the index list. If the list is long, efficiency will dictate that a hierarchy of directories be incorporated. The insertion and removal of terms from the active index list and the corresponding maintenance of directories should be entirely automated.

In cases where the subsumed index terms are from an infinite set, one does not generate them, but rather one examines the index list to see if it contains any index terms that could have been so generated. Clever coding and ordering of the index list can substantially shorten this examination. Actually, it may be profitable to introduce more than one index list or even one list for each class of index terms.

In general, one does not expect to retrieve on a single index term, but rather on combinations. In fact one might expect to perform arbitrary Boolean functions on sets of index terms. In addition, one may desire some external control over the selection of subsumed index terms for any particular term in some preestablished way. In the example above where one indexed all transactions of amounts larger than $1000, one may want for a particular retrieval only those transactions with amounts greater than $100,000. Each index term, therefore, should permit an operation with parameters that can control the selection of subsumed index terms. The permissible operations must be defined for each index list.

In many cases, one needs to relate index terms of different classes for retrieval. If one wants to retrieve all employees who earn $5,000 and who have 2 children, it is not sufficient to use the two index terms connected by "and" since one may get employees with two children earning other than $5,000 or employees earning $5,000 with other than two children if more than one person is described in the indexed record.

It is necessary, therefore, to group index classes and to let the terms in the grouped classes index portions of a record; such portions will be called subjects. For example, consider a document (Fig. 4) that mentions factory #1 and lists the names of all its employees together with the salary each earns and the

Factory #1			
Employee	Name	Salary	Children
1	Abel	$ 5,000	2
2	Baker	10,000	2
3	Carter	5,000	3

Figure 4. Tabular summary of a sample document.

number of children each has. Suppose in particular, there are three employees Abel, Baker, and Carter who earn $5,000, $10,000 and $5,000 and who have 2, 2, and 3 children respectively. If the document were given the number 100, one could index as illustrated in Fig. 5, where the index numbers may consist of one or two parts, the first referencing the document and the second referencing the employee in the document. Each person has become a subject within the factory; a sub-subject of the subject "Factory." Actually, if more than one factory were mentioned, one would require three parts for each index number, one each for document, factory, and person. Note that the number of persons per factory is unbounded. The grouping of index classes, which parallels the scheme for assigning index numbers and which parallels the structure of subjects in documents, is tree-structured.

DATA STRUCTURE

An information store and its interaction with indexing may now be described. The data structure is concerned with the storage of information collected on the various subjects of interest for a particular information retrieval system. The subjects, for example, may exhibit

$5,000—100,1
—100,3
$10,000—100,2
2 children—100,1
—100,2
3 children—100,3
Abel—100,1
Baker—100,2
Carter—100,3
Factory #1—100

Figure 5. Example of indexing structure.

hierarchical relationships such as persons employed by factories or components within assemblies.

The informational units for processing and for retrieval are fields. Fields may possess subfields or be combined to form larger fields. A field may exhibit three kinds of variability; the possible values of a field may not all have the same size representation, or more important for present considerations, a field may require a variable length list of values which are considered either as additional subfields or as alternative values called replications. The smallest fields are called primitive fields. Each primitive field has a specified ordered set of values called primitive values. Fields are defined recursively as being either a primitive field or a list of fields; these latter fields are then called subfields. A field value, or more briefly value, is correspondingly either a primitive value or a list of subfield values. If some subfield is replicated, there is not a uniquely determined value. Accordingly, the scope of a field, or more briefly scope, is defined as the set of all values obtained by ranging over all combinations of subfield replications. The set of all possible field values is ordered in one or more ways as extensions of the orderings defined for each subfield. For example, the field "Personality name" may have the three subfields, "Last name," "First name," and "Middle name," which are further broken down by "Characters" to permit such retrievals as "all electrical engineers in Detroit whose first name is Alan," or alternatively, whose last name ends in the letter "y." Note that a distinction is drawn between the field value "Smith" and the instance of the subject "Person" that is known by the name "Smith." Subjects are organizational entities in correspondence with the objects, individuals, or abstract categories to which the descriptive information pertains. It is convenient to also distinguish—from the fields which describe subjects—the fields, called auxiliary fields, which describe the relationship between other fields and the subject. For instance, the population of a city may be accompanied by a date and reference source. In any implemented system there will also be control fields to compensate for the several kinds of format variability allowed.

It is now possible to define *information* as the holding of a given set of relationships among a set of fields with given values. The informa-

tional content of a document is meaningful as the totality of information resulting from an analysis of the document in terms of field structure and values. An analysis of each document is required in a collating system since documents are not retained in the main data store.

Some fields are distinguished as the generators of index classes. An index class generated by a field consists of the set of all possible field values; these values are called index terms of the index class—e.g.:

Index class: Manufacturer-Vehicle
Index term: Ford-Truck

The set of index terms is organized into a number of directed linear graphs (not necessarily trees or lattices) by a number of functions on the set of index terms whose values are pairs of sets of ancestor and descendent index terms respectively.

Other fields are distinguished as the generators of subject classes. The terms subject value and subject scope are carried over from the field definitions. An attribute of a subfield which is also a generator of a subject class will be referred to as a sub-subject attribute. The members of a subject class are units of storage corresponding to the subject scopes actually retained in the information store of the system. Subjects are the units of indexing. Each index term points to a number of subjects, and each subject is pointed to by at least one, and usually more, index terms.

In the index list, besides identifying a particular subject, it is necessary to identify the class to which the subject belongs; this enables distinguishing between the "Location" in which a person resides and the "Factory" in which he works. The choice of subject classes is matched to the choice of index classes so that the set of subjects to which a pair of index terms jointly apply is precisely the intersection of the two sets of subjects to which the index terms individually apply. Consider for example, the subject class "Automobile ownership" consisting of the two subject classes "Family" and "Auto" (Fig. 6) and an instance of "Automobile ownership" (Fig. 7). The items with asterisks are subject identifiers and the subject "Auto" is replicated.

In an information retrieval system it is possible that only a small fraction of the information is deducible from the index list-subject

Subject—Automobile ownership
Subject—Family
  Name
  Number in family
  Number of drivers
Subject—Auto
  Manufacturer
  Year

Figure 6. Example of subject structure.

relationship. In the design of such a system, the statistics of processing and retrieval, the sizes of files, and the nature of the hardware configuration are prerequisite to making a reasonable determination of indexing structure and the organization of subject storage. In general, information will be stored in units of records which correspond to subjects (usually with sub-subjects) for which the statistics of record size are well matched to the storage media. For many applications the number of fields per record may be rather large, so that one would not want to reserve space in the record for fields which may never be given values in a particular record, and considering value replications, one cannot know in advance the number of field values to be stored in a given record.

Two elements of information storage, the field and the record, have been introduced. The primitive field is the smallest piece of data that will be named and manipulated. The record is the largest piece of data that will be manipulated as an entity. Records are composed of fields. Within a record, any field or combination of fields may be associated with other fields or

**Automobile ownership
 *Family
  Smith, John Q.
  5
  3
 *Auto
  Ford
  62
 *Auto
  Volkswagen
  59

Figure 7. Example of data structure.

combination of fields provided the association is independent of the particular datum that "fills" the field and provided the associations can be arranged to represent a tree structure for the record. A field is a form that has a value. The value may be any of a set of terms defined by the application or it may be the vacuous symbol. A record is a tree structure of fields independent of the values of the fields. It is to be emphasized that the values for a given field are not required to be tree-structured. It is only the fields themselves that must be so structured in order to provide a scheme for assigning names to data sets stored in the record.

Any piece of information in the system has a name which is factorable into:

Record identification
Subfield replicate
•
•
•
Subfield replicate
Primitive field replicate

The derivation of names follows directly from the description of records, subjects and fields. It is recognized that a given term might possess different descriptions for input, indexing, storage, processing, and output as demanded for processing and storage efficiency.

RETRIEVAL

When one submits an interrogation saying he wants all information about a particular hurricane, he is saying that he wants to restrict himself to documents about that hurricane. Actually, information is retrieved. The documents from whence the information derived may be retrieved to substantiate a claim or to furnish more detail than is formalized within the system. Similarly, if he wants information about all factories that have employees who earn $10,000 and who have 2 children and that have employees who earn $5,000 and have 3 children, he is likewise restricting himself to certain records. The terms that do the restricting, we call restrictors. Restrictors need not always be index terms. For example, one may not choose to use the date of a document as an index term, but one may still choose to restrict by that date. This assumes that one can find such a date in a given record and test it. This is no problem in an information retrieval system, but it requires some special organizing in a document retrieval system.

In an information retrieval system, one may have many restrictors that are not index terms. These terms must be tested after the appropriate record is selected via the index list. This is essentially a two-stage selection procedure. First using the index terms, one gets all records that are associated with the index terms and their subsumed terms. The choice of which restrictors to use for this purpose and the order in which to use them is not necessarily prespecified. This may be determined for each retrieval by applying heuristics based on previous retrieval efficiencies. The subjects obtained are then tested, and only those are chosen which satisfy the remaining restrictors.

In a document retrieval system one displays the whole document that has been retrieved and it is up to the interrogator to find what he wants. In an information retrieval system, one may specify those items (the extractors) he wants to see. In non-collating systems the extracted information must be culled to remove redundant and inconsistent information. It is necessary that the interrogator know what items it is possible for him to see; that is to say, he may extract only those items that can be recognized by the system. This, however, is nothing new, for the same requirement is imposed upon him in using restrictors. The extractors can be expressed in the same manner that the restrictors are expressed. If information is not present, the extractors or groups of extractors may be ignored.

With a little additional effort, one may extract only if the sought data is subsumed by an index term. For example, one may extract a location only if it is subsumed by "Michigan." This is not quite the same as a round-about restrictor. A more vivid interrogation that exemplifies this is: "Give me (extract) the names and addresses of (restrict) all persons in factory #1; also give their positions if (conditional extraction) they are among the professional staff." Without the conditional extraction, one could at best obtain two lists—one with the names and addresses of all employees and one with the names, addresses and positions of the

professional staff. Of course one can permit Boolean functions of extractors.

In collating systems, once one has a set of records to be displayed, there is the problem of sorting them in some specified manner. If the field being sorted on appears at most once in every record, there is no new problem. Suppose, however, each record from which we have extracted our data has as subject a particular apartment house in some city and suppose that among the data in the record are the names of all the tenants and the address of the apartment house. If the interrogation reads: "In alphabetical order, list the names and addresses of all people who live in apartment houses in that city," artificial records are created for sorting with the address of the house duplicated for each person who lives in it. Further complications arise if more than one level of sorting is specified at one time.

The retrieved information is extracted as subsets of selected subjects. This information therefore has subject structure and must be printed so as to reflect that structure. That is, replications of a particular sub-subject or field are vertically aligned and are vertically spaced so as not to conflict with each other. This also applies to values and repeated fields that exceed allotted column widths.

PROCESSING

Among the information processing programs are three which characterize solutions of the information retrieval problem. The first program prepares an information structure which represents the terms and syntax of a document or interrogation in a form convenient for machine processing. The same language is used for couching requests and for preparing inputs. A thesaurus is used to identify terms, to resolve some problems of synonymy and ambiguity, and to associate general and specific meanings. The variety of inputs renders impractical a fixed data structure to accommodate any allowed input. Different classes of terms require different sets of programs to incorporate a term into the information structure, and individual terms may introduce processing variations.

The second program extracts from the data store information structures satisfying selection criteria. The types of field structures and the selection criteria will determine the subprograms to be used for selection and the amount of working storage required.

The third program forms composites of information structures. A number of information structures related to a given one are selected as candidates for merging into one or more composites. This collating process may discover conflicting information which is then directed to programs which resolve the conflict. There is no inconsistency in retaining conflicting information. As additional related information enters the system, the conflicts will be reexamined until a decision can be made to resolve the conflict. Many programs are required to handle the many different situations that may arise. These programs are large and have dissimilar storage requirements.

The computer programs are large because the design problems do not admit easy solutions. The data is varied; the processing is intricate, and the programming is correspondingly complicated. Each option and each event considered adds to the sizes of the programs. The computer programs may be large and yet the result may be only a crude approximation of the desired intelligent behavior. More cases, more flexibility, and new levels of processing detail may all be required in order to achieve a fully useful result. Finally, the programs are large because the problem is large and changing; or rather, the programs are large because there is not a single problem but a series of problems arising out of changes in requirements, changes in technology, and changes in understanding the problem—throughout, the programs must remain responsive to change.[10]

By a large program is meant first one which translates into such a large number of machine instructions that the program cannot fit all at once into the main memory of the computer. But the size alone is not a sufficient criterion since the program might be organized to consist of a number of sub-programs which may be brought into the main memory in a simple sequential fashion. In order for largeness to be an inherent characteristic of the program, mere size must be related to program complexities.

Program complexities may arise from both the organization of the computational algorithm and the manner in which the program is executed. The flow of the code into the main memory of the computer may be complicated

by requiring the programming parts to be executed in a variety of arrangements; these arrangements may be determined by the computer data being processed. The execution of a code may be interrupted and the code removed from the main memory in order to allow another program to proceed; at a subsequent time the program is restored to memory in such a way as to resume the execution from the point of interruption. The program may instruct the computer to perform an elaborate computation or the data may possess intricate relationships; in the latter case the literature on list organization and list processing is relevant to information processing.

This brief excursion through representative processing was conducted to suggest:

1. That many programs are involved;
2. That the next program to be executed is a function of the information being processed;
3. That the programs have variable storage requirements.

The storage requirements of the programs are such that only a few of them may coexist in the computer's memory. Accordingly, one expects the proportion of input-output transmission devoted to programs as opposed to data to be high; perhaps two orders of magnitude higher than for representative data processing problems. The managing of program flow becomes a major part of the information retrieval problem.

PROGRAMMING SYSTEMS

Consider for a moment the development of data processing. Faced with mountains of data, first machines and later computers and computer programs were developed to order and maintain the data; eventually, generalized file control programs appeared, not to process data, but to merely transmit data to and from data processing programs. Now, faced with mountains of data and smaller mountains of programs, additional programs are needed to manage the programs which control and process the data. These second level programs constitute what is commonly referred to as an *executive system.*

As processing becomes more complicated and executive systems become more deeply enmeshed in the computation, the programs which make up the computation must adhere to more and more conventions. Before the point is reached at which the conventions become overburdening to the programmer, some of the work of following the conventions is passed over to compilers and other automated programming aids. Now if a compiler is to produce programs to be executed by a given executive system, the compiler is constrained by the executive system and they become part of a *programming system* which encompasses the design, construction, debugging, maintenance, and operation of programs. [1, 5]

Returning to the management of programs for the information retrieval problem, there is an initial requirement for an executive system which interprets transfers between programs to insure that the program transferred to is in memory and, in case it is not in memory, to bring it into memory after saving the necessary registers. It must be noted that programs are not disjoint; in general, successive programs will have considerable overlap of common code and working areas. The definition of *program* must therefore be broad enough to include portions of code produced by compilations of different programs. Furthermore it is desirable to allow assigning work areas to programs as their requirements become known during execution.

An example will suggest the richness of program structure that is applicable to the information retrieval problem. A program in use occupies an interval in space-time. The spatial entities are memory sequences, input files, output files, other programs which this program might call into use, and such entities borrowed at execution time from programs already in use. Each entity may be further partitioned to take advantage of hardware configurations. For instance, a memory sequence may be partitioned to take advantage of non-contiguous memory locations; likewise a magnetic tape file may be partitioned into reels by physical necessity or to permit faster accessing. At the program execution level, time is single valued. Time is partitioned into execution phases during which some subset of the spatial entities are in use. As the execution progresses from one phase to the next, some spatial entities drop out of use and others come into use. The term *phase* refers to the spatial entities used during an interval of time; the spatial entities used

during a phase may be used again during a later time interval; phases are ordered with repetition by the computation to form *execution time.* Program execution time is organized by parallel processing and multi-programming techniques to form *process time.*[2] Thus as applied to the entire program complex, time has a multi-valued character. At any given moment of *computer time,* calculation has proceeded to some point in some phase of each process. Concurrently, computation is proceeding in one process, input-output transmission is going on in other processes, and the remaining processes are waiting in various states of readiness. These two components of time are controlled by different mechanisms. Process time is changed in response to hardware or external signals; phase time is changed in response to computation.

There is no intended implication as to how or when spatial entities are assigned to hardware facilities. The above remarks apply to those problems in which all assignments can be made by the programmer and compiler and to those problems in which some assignments are postponed to the time of loading and to the time at which they are required by the computation.[6, 9]

* * *

The author is indebted to many persons for conversations which have contributed to the developing or testing of these ideas. Some have left their imprint upon the paper and are singled out for special acknowledgement: Anatol Holt for the introduction of phase and sequence into program structure, John Goodroe for the connection with multi-programming, and Bob Colilla for developing the sections on indexing and retrieval.

BIBLIOGRAPHY

1. CHEATHAM, T. E., JR., COLLINS, G. O., JR., and LEONARD, G. F., "CL-1, An Environment for a Compiler," *Communications of the Association for Computing Machinery 4,* January 1961, pp. 23-28.
2. CODD, E. F., "Multiprogram Scheduling," *Communications of the Association for Computing Machinery 3,* June-July 1960, pp. 347-50; 413-18.
3. COLILLA, R. A., and SAMS, B. H., "Information Structures for Processing and Retrieving," *Communications of the Association for Computing Machinery 5,* January 1962, pp. 11-16.
4. GURK, H., and MINKER, J., "The Design and Simulation of an Information Processing System," *Journal of the Association for Computing Machinery 8,* April 1961, pp. 260-70.
5. HOLDIMAN, T. A., "Management Techniques for Real Time Computer Programming," *Journal of the Association for Computing Machinery 9,* July 1962, pp. 387-404.
6. HOLT, A. W., "Program Organization and Record Keeping for Dynamic Storage Allocation," *Communications of the Association for Computing Machinery 4,* October 1961, pp. 422-31.
7. MILLER, L., MINKER, J., REED, W. G., and SHINDLE, W. E., "A Multi-Level File Structure for Information Processing," *Proceedings of the Western Joint Computer Conference,* 1960, pp. 53-60.
8. MINKER, J., "Implementation of Large Information Retrieval Problems," Gordon Research Conference, New Hampton School, New Hampton, N. H., July 1961, 9 pp.
9. SAMS, B. H., "Dynamic Storage Allocation for an Information Retrieval System," *Communications of the Association for Computing Machinery 4,* Oct. 1961, pp. 431-35.
10. ———, "Some Observations on the Development of Large Programs," First Congress on the Information Sciences, Hot Springs, Va., Nov. 1962, 28 pp.

AN OUTLINE OF THE REQUIREMENTS FOR A COMPUTER-AIDED DESIGN SYSTEM

Steven Anson Coons
Mechanical Engineering Department
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

HISTORICAL

In the early 1950's at M.I.T. the Servomechanisms Laboratory (now the Electronic Systems Laboratory) devised and developed the first automatically controlled milling machine.[1] The controlling information for the machine was introduced in the form of punched paper tape, on which all dimensional information and instructions for the various feeds and cutter speeds was contained. At first the punched paper tape was prepared manually by some human operator who translated, in effect, the detail drawing of the part to be machined into numerical form and then into appropriate patterns of holes in the tape. This was a tedious and entirely mechanical chore, and it was only natural that short cuts in the process began to suggest themselves. The scope of such short cuts began to spread through the fabric of the technique, and it was not long before the computer was involved in implementing them.

In the late 1950's the Computer Applications Group of the Electronic Systems Laboratory developed in great detail a complete, automatic system for preparing these punched paper "director" tapes from detail drawings. To be sure a human operator was still required, but the process of converting a drawing into tape was very much simplified, and the combination of the Automatically Programmed Tool or APT System[2, 3, 4] with numerically controlled machine tools has subsequently proven to be of significant economic importance in many industries, notably in the aircraft and missile industry. The APT System has gone through a vigorous history of rapid improvement and development, partly at M.I.T. and partly with the active participation of industry. Currently maintenance and development are being carried out by the APT Long Range Program at the Armour Research Foundation with over 25 companies participating.

About four years ago there was a meeting of members of the Computer Applications Group with members of the Design Division of the Mechanical Engineering Department to see whether it might be possible to take another important step. At that meeting we discussed the possibility of using the computer in a much more direct and powerful way in the chain of events that begins with the original concept as envisioned by the design engineer and culminates in the production of the finished device. We outlined at that meeting a system that would

This work has been made possible through the support extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory by the Manufacturing Technology Laboratory, ASD, Wright-Patterson Air Force Base under Contract No. AF-33(600)-42859. It is published for technical information only and does not necessarily represent the recommendations or conclusions of the sponsoring agency.

in effect join man and machine in an intimate cooperative complex, a combination that would use the creative and imaginative powers of the man and the analytical and computational powers of the machine each with the greatest possible economy and efficiency.

We envisioned even then the designer seated at a console, drawing a sketch of his proposed device on the screen of an oscilloscope tube with a "light pen," modifying his sketch at will, and commanding the computer slave to refine the sketch into a perfect drawing, to perform various numerical analyses having to do with structural strength, clearances of adjacent parts, and other analyses as well. Based on such analyses the designer would modify his original design concept, and again call for an analytical procedure by the computer. In some cases the human operator might initiate an optimization procedure to be carried out entirely automatically by the computer; at other times the human operator might intervene, as he might do for instance if in a certain iterative process he observed the computer laboring fruitlessly to satisfy mutually incompatible constraints unwittingly imposed, or attempting to find a solution to a problem in a mathematical region which might seem to the computer a likely place to look, but which to the man might be obviously far afield. The different powers of man and machine are complementary powers, cross-fertilizing powers, mutually reinforcing powers. It is becoming increasingly clear that the combined intellectual potential of man and machine is greater than the sum of its parts.

Since this meeting, a formal arrangement was created for the combined efforts of the Computer Applications Group and the Design Division to work together in a broad study of what we call Computer-Aided Design. This activity is supported by a contract from the same Air Force group that sponsored M.I.T. efforts in both numerical control and APT. The investigations under the contract range over the entire spectrum of computer technology and of design philosophy and methodology. Out of the investigation will come the design for a man-machine organism to accomplish the design process in a way far easier than has ever before been possible; but as by-products will come new computer techniques and an enriched understanding of the creative thought process.

THE DESIGN PROCESS

The design process begins with a graphical description of a proposed device or system to satisfy a human need. To say that the description is graphical is to assert that at the very inception of an idea the designer's understanding of his creation is almost visceral instead of intellectual. He perceives his idea at first not in the perfection of a well-turned English word description, nor in the precision of a mathematical formula, but in some nebulous assembly of building blocks of structure, vaguely beheld; he "feels" his creation. The sketch forms the natural bridge between these vague stirrings of the imagination and the subsequent precise statement of the refined details of the concept.

At this early stage, decisions to keep, to modify, or to discard part or all of the original concept are made in a qualitative way, based upon qualitative criteria. The modified concept leads to further qualitative decision making, and to further modification of the concept. While this is going on, the concept which was at first nebulous and incomplete begins to assume a more concrete solid character; it becomes better defined, until at some stage it is well enough defined to permit more precise analytical tools to be applied.

At first such analytical processes are very simple; the mathematical modeling is crude, and the actual calculations need not be carried out in great detail, nor to very great numerical precision. These calculations again lead to modifications of the concept and subsequently to more precise analysis. It is typical of the design process that such iterations—from concept, through analysis, evaluation of the analysis, decision to modify the concept, and finally to a new concept—form loops that are traversed again and again, until eventually the designer judges the design adequate to satisfy some scale or scales of value judgments.

In the design process, the designer is concerned with a large set of variables, some continuous (like the weight of a part) some belonging to discrete "point sets" (like the material: steel, brass, lead, plastic.) Moreover, these variables are interrelated, or cross coupled, in a very complex way. Some of the cross couplings are weak, some are strong. If the relationships happen to be linear, the cross couplings are constant in strength, but usually

the relationships are non-linear, and the mutual influences of the various variables change with their values.

The designer structures such relationships so that he can thread through them, taking advantage of the loose couplings where possible, to obtain hopefully an exact, but more usually a first, or second, or closer approximation to the values of the variables. It is not at all unusual for this structuring to be done graphically, in the form of block diagrams or linear graphs or information flow charts. Thus he uses a graphical form for both the topological and geometric description of the design, and also for its abstract description in terms of physical function.

At the conclusion of the design process, the final result must be carefully defined so that it can be built. This is the function of layout draftsmen and detail draftsmen. If automatically controlled machines are involved in the fabrication processes, programmers are also a part of the system.

When we look at such a design sequence we see a few engineers performing highly creative tasks at the beginning, coupled with a very large number of draftsmen and technicians who perform relatively uncreative tasks over a fairly long period of time. Some of these tasks require high degrees of intellectual effort, such as stress analysis or aerodynamic analysis, but they are none-the-less not in themselves of a creative nature (except in those cases where new mathematical techniques are designed and put to use). Other tasks are obviously of a purely mechanical nature; for example, a detail draftsman does nothing creative whatever. At the worst, he merely traces the outline of a part from the layout drawing, and adds the dimensions. Usually this drawing goes directly to some machinist or patternmaker in the shop, but sometimes it is used by a part programmer and converted by him into symbolic information for use by a computer to prepare punched tape for automatic fabricating machinery. These are all essentially mechanical operations, however, and it is quite clear that at least in principle, the computer can be made to deal with them all.

COMPUTER SYSTEM REQUIREMENTS

A computer system, to work in partnership with a designer, must have several clearly definable capabilities. It must be able to accept, interpret, and remember shape descriptive information introduced graphically. When such a graphical input capability is properly designed, the man-computer combination can manipulate the elements of a drawing in an entirely new way, with a freedom and precision far surpassing what is possible with pencil and paper.

Beyond shape description, such a graphical facility should be an extension of language in general. It should be possible, as has been indicated earlier, to use such a graphical mode to structure abstractions. This has been brought out with great force and clarity by Engelbart,[5] where he remarks in effect that the essentially one-dimensional nature of symbolic language is not wholly adequate to exhibit the interconnections of ideas.

Coupled to this graphical facility must be a computational facility for unravelling and performing all of the mathematical analyses and computations that pertain to the design process. These lie in the fields of stress analysis, aerodynamics, thermodynamics, electrical network analysis, fluid dynamics, and many others. The computer should also be able to furnish information about standard parts, standard materials, and standard processes. This is essentially an operation of catalogue storage and retrieval.

There are two quite different philosophies of approach to the achievement of these aims. One approach would be to imbed in the computer a large set of special purpose packaged processes, each designed to perform some special task. If the assembly of such a library of special routines could be made complete enough, then the system would exhibit to the user on the outside an appearance of complete flexibility and generality. This would be satisfactory so long as the designer never called for a capability not already rigidly imbedded in the mechanism. But the design process is unpredictable. Indeed part of the design process consists in designing new ways to perform the design function itself. This is a higher order of design activity, a sort of meta-design (like meta-mathematics) that clearly is outside the scope of any rigid set of special processes that can be anticipated at the beginning. This very real consideration leads quite naturally to the

second philosophy, in which the large population of special purpose routines is replaced by a few (perhaps indeed only one) routines of the utmost generality, so designed that it permits of its own modification by the designer, using his own natural language forms, including as we have said, the graphical form.

The Computer-Aided Design System should be capable of carrying on conversations with, and performing computations for several designers at several consoles substantially all at once. In this way each designer can be immediately aware of what the other designers are doing, and thus avoid one of the truly severe problems of intercommunication that designers face today.

The flexibility and ease of communication with the computer will encourage the designer to use more detailed and more accurate mathematical models of the real physical system than he has been willing or able to use in the past. This will result in a more rapid and a surer approach to an optimum design and to a design that may be relied upon, especially in those very new areas where only meager or fragmentary experience has been accumulated from past designs.

It will be possible to design at an exponential rather than a constant rate, because sub-elements of a design, once constructed, will be available on command in their entirety, and can either be incorporated as they exist or can be modified at will. On some far off day it may even be possible to call up last year's automobile on the oscilloscope, to wave the magic wand of the light pen, and in a very short time to create the modified new version from the old. This will be, in a sense, a mechanization of experience.

It will be possible to observe the actual moving action of a mechanical device, or the varying currents and voltages in an electrical device, rather than the static, frozen, time sections of these motions and currents and voltages as we must now do by present methods of analysis.

Finally, the system will be so general that it will be applicable to any creative activity. For example, the *general* problems of the architect, the machine designer, and the electronic designer are the same, but the *specific* details of their problems bear scant resemblance one to another. Yet an appropriately designed system will be so flexible that it will enable each discipline to modify the structure to fit its purpose.

There is considerable evidence that our intellectual tools influence to a very great extent the form and scope of our intellectual works. It is quite certain that when the computer replaces pencil and paper in this very real way, it will bring about a truly miraculous change in man's intellectual potential.

PRESENT CAPABILITIES

We have already come quite a way toward accomplishing some of the desired ends as outlined. The writer, using Sutherland's Sketchpad Program[6] on the TX-2, has set up and solved five separate engineering problems in the course of a few hours. It is an extremely flexible and versatile means for communicating with the computer in a graphical language.

Sitting at the console of the TX-2, the writer constructed a geometric figure which represented the cubic algebraic polynomial. The details of this figure are not important; the principle is quite simple and very general. It is easy to construct such a figure for polynomials of any degree, and for both real and complex values of the coefficients and of the variables. Once constructed, the x variable can be manipulated with the light pen, and the resultant value of y is automatically obtained. This happens by virtue of the computer's ability to satisy the geometrical constraints imposed by the figure.

The second problem was the construction of the general second degree curve based upon a purely geometrical construction, and the third problem was one with which Sutherland had already experimented. A pin-jointed structure is drawn, certain points are fixed, loads are applied, and the deflections of the structure are then automatically calculated, and exhibited on the screen, along with the numerical values of the percentages of elongation of each member.

Fourth, a kinematic linkage was drawn, the driving link was rotated, and the motion of the connected links could then be observed. The linkage could then be modified to yield desired changes in its behavior.

Fifth, a region was drawn in which two-dimensional flow of an ideal fluid was to take place. By invoking the principles of graphical field mapping, the computer adjusted the stream lines and equipotential lines of the field so as to give a good solution to the problem.

These problems are all substantially very different. No special computer program written to solve one of them is of the slightest use in solving any one of the others. But the great range of applicability of generalized constraint satisfaction makes it possible to solve them all. Admittedly special purpose programs can be devised that will solve some of them more efficiently and rapidly than these graphical methods. But on the other hand, the graphical solution is in some cases the most efficient method known. In such cases, since the computer can perform graphical manipulations at least a thousand times faster than a man, such techniques are still highly economical.

Many parts of design are well-enough understood and of general enough utility to warrant special programs. A mixture of general and special techniques is most appropriate for full Computer-Aided Design. Using a different computer, the IBM 709 at the M.I.T. Cooperative Computer Laboratory, together with a special purpose program, it is now possible to draw a cantilever beam on the screen, and to type in its precise length on the flexowriter. Then the vector loads are drawn, and their magnitudes are typed in on the flexowriter. Finally, a section of interest in the designer is drawn, and on this section a particular point of interest is indicated. The computer hesitates for a moment, and then types out on the flexowriter the bending stress, the axial stress, the transverse shear stress, the torsional shear stress, and the combined stress at the point of interest. This is a special purpose program, but its general applicability and efficiency make it well worth while.

In addition to using the generalized constraint satisfaction for computation, the primary purpose of a system such as Sketchpad is graphical communication. In Sketchpad III, as reported in the paper by Johnson,[7] we have a means for drawing and manipulating figures in three-dimensional space. This is of course essential to the designer of mechanical or structural devices and objects. All of the capabilities of the two-dimensional version of the graphical input are implicit in the extended system.

The work on the theory of language and operators described in the following paper by Ross and Rodriguez[8] is directed toward making direct communication with the computer not only possible, but easy, while at the same time enabling the computer to remember and manipulate the many complex details implied by general statements. Then, using these capabilities, the designer at the console can communicate not only the problem itself, but also the problem solution structure, either generalized or specialized, in whatever form and in whatever language is appropriate, meaningful, and most efficient.

CONCLUSION

The historical section of this paper describes our original broad concept of the Computer-Aided Design System. This concept was even at the beginning formed on fairly grand proportions, and it is encouraging that even after the intervening time, its form has not been substantially changed. We have not relaxed our objectives, and as we see the details of the broad framework filling in and taking shape, we are encouraged to believe that however ambitious it might have seemed in those days, our results indicate that practical Computer-Aided Design will indeed, in some not too far distant future be a reality.

BIBLIOGRAPHY

1. PEASE, W., "An Automatic Machine Tool," *Scientific American*, Vol. 187, No. 3, pp. 101-115, September 1952.
2. ROSS, D. T., "Automatically Programmed Tool System," pp. 59, 60. "The APT Joint Effort," p. 70, *Mechanical Engineering*, Vol. 81, No. 5, May 1959.
3. WARD, J. E., *Automatic Programming of Numerically Controlled Machine Tools, Final Report.* Report 8753-FR-3, Electronic Systems Laboratory, Massachusetts Institute of Technology, January 15, 1960.
4. BATES, E. A., "Automatic Programming for Numerically Controlled Tools—APT III," *Proceedings of the 1961 Computer Applica-*

tion Symposium, The Macmillan Company, New York, pp. 140-156, October 1961.

5. ENGELBART, D. C., "Augmenting Human Intellect: A Conceptual Framework," Report SR-3223, Engineering Sciences Division, Stanford Research Institute, October 1962.
6. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Communication System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume).
7. JOHNSON, T. E., "Sketchpad III, A Computer Program for Drawing in Three Dimensions," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume).
8. ROSS, D. T., and RODRIGUEZ, J. E., "Theoretical Foundations for The Computer-Aided Design System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume).

THEORETICAL FOUNDATIONS FOR THE COMPUTER-AIDED DESIGN SYSTEM

Douglas T. Ross and Jorge E. Rodriguez
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

I. AN APPROACH TO THE COMPUTER-AIDED DESIGN SYSTEM

A Computer-Aided Design System for general use must have a unique and powerful organization. Even the simplest of design problems involves the exercise of many disciplines and the carrying out of many types of activity. Since the area of applicability of the design system is to be essentially unlimited, we know from the beginning that the system itself must be very large and complex. Even though only a few of its features may be exercised on any given design problem, there is no way of predicting which portions of the system will be required nor how they will be used. Furthermore the designer or engineer who is using the system cannot be expected to be a computer programmer, and it must be possible for him to carry out his design function in a way which is natural to him, and without his being aware that the statements and actions that he performs are in fact constructing and executing large numbers of highly complex computer programs. Although to be sure the user must learn and become facile with the basic vocabulary and manipulations of the system, the system must be so designed that he finds his normal thought processes aided, augmented, and stimulated by the use of the system in such a way that he is able to think almost entirely at the concept level within his own field of interest, while at the same time carrying out data processing activities of extreme complexity.

A. *Necessity for an Evolutionary Approach*

The broad requirements of general applicability and extreme flexibility of use preclude the possibility of constructing a Computer-Aided Design System by assembling a potpourri of specialized languages and computing systems under the control of a single grandiose executive routine. There are a great many existing specialized languages and programming systems for many of the individual areas which must be covered by the Computer-Aided Design System, but each of these languages and systems has its own restrictions and interwoven computational complexities so that it would be completely impractical to attempt to integrate such systems in a straight-forward manner. Furthermore, such a brute force approach would not satisfy the Computer-Aided Design requirement in the first place, since there would be little or no cross fertilization between the various systems, even if the mechanics of translating data and control information from one

This work has been made possible through the support extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory by the Manufacturing Technology Laboratory, ASD, Wright-Patterson Air Force Base under Contract No. AF-33(600)-42859. It is published for technical information only and does not necessarily represent the recommendations or conclusions of the sponsoring agency.

system to another could be solved in a moderately satisfactory manner.

Instead of becoming discouraged by the immensity and impracticality of such an approach, and limiting our expectations for the system to a set of requirements which could possibly be handled economically in that way, we seek instead to find an approach which will attack the problem with more finesse and provide a system with the full desired potentiality. The first step in this direction is to recognize once and for all that it is completely impossible to construct a system which will satisfy the requirements immediately and without modification. In fact to postulate the existence of a closed system for Computer-Aided Design as we mean it is completely and absolutely contradictory to the very sense of the concept.

Since the blatant denial of the possibility of creating a closed system which will satisfy the requirements is so central to the approach to be taken, we will dwell on it a moment. The contradiction stems primarily from the fact that if the system has, built in, a certain way of accomplishing a task, but the individual user does not wish to perform the task in that manner, then unless it is possible for the user to substitute his own way of doing things for the way that is already built in, the system will not satisfy the basic requirement of naturalness and ease of use. Another basic source of the contradiction, and one that may seem to some to carry more weight than the requirement that the system be adaptable to the arbitrary whim of an arbitrary user, is that it is inconceivable to build in beforehand all possible solutions to all possible problems. Thus the very nature of the system must be such that its area of applicability is continually extended by its users to provide new capabilities as the need arises.

Thus we see that the basic thing to be provided in the initial organization and structure of the Computer-Aided Design System is a capability for growth, expansion, and modification. Whereas it is indeed contradictory to consider satisfying the requirements with a closed system, the concept of an evolutionary system is not only in harmony with the requirements, but in fact makes those requirements more meaningful. If, in fact, the system can be so organized that it can naturally be molded to suit the needs and interests of individual users, then the concept of a general Computer-Aided Design System not only begins to seem possible, but practicable as well.

B. *Plex Structures for Problem Modelling*

If the points which have been made above concerning general applicability, naturalness of use, and expandability are considered, it becomes apparent that before anything else we must provide for a completely general method of storing and manipulating arbitrarily complex information from any source, and a powerful language facility for describing data forms and the desired manipulations of data. It almost goes without saying that if all of the necessary information about a problem and its method of solution can be described then the problem can be solved. The stringent requirements under which we are operating merely put an added emphasis on the need for a single *unified* approach, in order that all of the many features required for a practical Computer-Aided Design System can be cut from one cloth.

These considerations led to the concept of "*n*-component elements" and "plex" structures, which have been described elsewhere.[1, 2] Stated briefly, an *n*-component element is a single unit of information about a problem, which specifies in each of its components one attribute or property of the element. There may be any required number of components in an element, and each component may be considered to be a pointer which indicates the specific property. Numerical or metric properties are interpreted as pointers to an appropriate measuring scale. Other components point to additional elements whose components give additional properties. The resultant structure of many elements pointing to each other and to appropriate measuring scales is called a *plex*, so that a plex is "an interconnected set of *n*-component elements."

An example of a very simple plex is the structure which specifies a line in two-dimensional coordinates, as shown in Figure 1. The line element contains a *type* component which

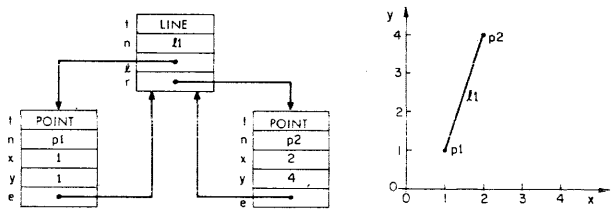


Figure 1. Example of a Modelling Plex.

specifies that it is a line, a *name* component, and two additional components which point to its end points. The point elements in turn have *type* and *name* components, and two additional components which specify the x and y coordinates of the point in some coordinate system, and also a component which points to the line element to indicate the endpoint relationship.

Note the precise correspondence between the parts of the plex and the problem of "lineness." The plex for line is in effect a *model* of the line, and everything associated with the concept "line" can be found in the plex either directly, or through computation. Thus, for example, the length of the line is not shown explicitly (although it easily could be), but may be determined from the coordinates given. Another fundamental concept of plex modelling is that only those components need appear which are of interest. Thus if only topological questions are to be considered with respect to lines (as perhaps in electronic circuit diagrams), then the components of the elements which contain the coordinates of the points could be omitted. Note, however, that omission of the coordinates shows that a different "line" concept is involved.

In general there is no unique set of components which represent a given concept, since it may take on new attributes or aspects depending upon how it is to be used. In any case the method of storing whatever properties or attributes are required is the same, and all of the structuring and sub-structuring of the data is explicitly displayed in the plex structure so that any required data is immediately available.

As was indicated above, the meaning or interpretation of a given component depends upon how that component is to be used by algorithms which perform computations or manipulations on plex structures. Just as it is impossible to conceive of an algorithm or program without data, data in turn is meaningless without something to use it. Therefore, the common conceptual framework which underlies all of the developments toward the Computer-Aided Design System is the concept of appropriate plex structures and processing algorithms for any required task. The contents of components of elements are obtained by writing A(B) to obtain component A of element B. More complex operations requiring several layers of nested sub-components are obtained quite naturally by writing an extended "referent," A(B(C(D))), which is read "A of B of C of D," so that the data acquisition aspect of all algorithms is uniform.

C. *The Role of Language*

The concept of a plex is philosophically satisfying and quite clearly is a very basic foundation stone leading toward the Computer-Aided Design System. It is difficult to conceive of anything simpler in basic structure, and yet quite obviously any collection of knowledge or information about something can be modelled to any degree of detail by an appropriate plex. It is also clear, however, that even seemingly simple concepts will obtain a very elaborate structure when all of the necessary inter-relations between subparts are explicitly exhibited by means of pointers and elements, and this leads naturally to the next major topic to be considered—how are plexes (which may contain thousands of pointers with many different meanings) to be constructed? In even simple applications plex structures become so elaborate and interwoven that they are almost impossible to unravel, and it is quite apparent that although they are the most natural way to store explicitly all of the details about a problem inside the computer, they are not at all well suited for human consumption.

The necessity for using very elaborate plex structures inside the computer to encompass all of the required areas of applicability, with the recognition that they are very inappropriate for human use, brings out forcefully the paramount importance of language to the Computer-Aided Design concept. Whereas the computer's "understanding" of a problem takes the form of a huge and complicated plex structure, the human's understanding of the same problem must be accumulated in his memory of the meaning of all of the statements which have been made about the problem, and which have led to the growth of the plex structure in the computer. In actual fact, the computer's view of the problem must match exactly that of the human, so that the human need not be aware of the internal structuring which allows the computer to understand the problem, but need only be concerned with the linguistic interpretation which is natural to him. The modelling plex must match exactly the meaning of the statements.

The problem with which we are left, then, is how to go from *language,* which is natural for the human, into *plex* structures, which will contain all of the required meaning of statements made in the language. If this major task can be accomplished—if we can go from the meaning of language statements (i.e., the human's understanding of that meaning) into the appropriate plex structure (i.e., the computer's understanding of the same meaning)—then we will have accomplished the first major step toward the full Computer-Aided Design System, since we may then begin from very elementary beginnings, and use the evolutionary expansion concept to incorporate any desired features into the system.

In the following section we describe and illustrate the first step in this process, namely the transformation of an input statement in a natural language into an appropriate plex form, called the *first-pass structure,* which will explicitly exhibit the syntactic and semantic content of that statement in order that further processing on the meaning of the statement can take place. The first-pass structure models the full concept of the statement, but not of the problem itself, in general.

The transformation of the first-pass plex into other plex structures representing the problem to which the statement refers, and an indication of the kinds of techniques which are appropriate for manipulating first-pass structures are presented in subsequent sections, which also illustrate the equally important converse problem of transforming information from a modelling plex into language statements.

II. OUTLINE OF THE ALGORITHMIC THEORY OF LANGUAGE

We may now turn to the subject of language itself. Since language (which includes graphical and mathematical, as well as verbal forms), is used to express all of our thoughts from the most mundane to the most elaborate and abstract, it is quite apparent that linguistic structures can become extremely large and complex. Our starting place for the consideration of language—our primary contention from which all else follows—is that large and complex linguistic structures, and in fact large and complex structures of any sort, do not arise cataclysmically and all at once, but are instead built up step-by-step out of simpler structures. We contend that this step-by-step growth process obeys discoverable natural laws quite similar to the physical laws of nature, and that if these laws can be formulated mechanically, then the grand complexity which we know without question must be the end result will arise naturally and of itself, as an immutable consequence of the action of those laws. Although we are still very much at the discovery stage concerning these laws, especially with regard to arbitrary complex structures, nonetheless sufficient progress has been made in the area of linguistic structures to bring home the point forcefully and engender considerable confidence that the basic approach is sound.

In the linguistic area, the concrete results which have been worked out rigorously and in detail thus far are presented elsewhere in the first introductory paper on the Algorithmic Theory of Language.[2] In the present paper we present a general tutorial outline of the theory and a somewhat popularized over-all description of its methodology and results in an effort to show how the Algorithmic Theory of Language supplies a fundamental foundation stone for the development of a Computer-Aided Design System of significant capability. For a full treatment of the theory the reader is referred to the definitive paper.

A. *Growth of Complex Structures*

A language is made up out of two kinds of things: vocabulary words and symbols. *Symbols* (which may be considered the nouns of the language) are the finest units of meaning and represent the atomic level beyond which there is no further substructuring. *Vocabulary words* on the other hand, have connective properties and can link together other words or symbols to make more elaborate non-atomic structures. In terms of our previous discussion, words and symbols are considered to be n-component elements whose components show the various properties which each element has. For theoretical simplicity we can without restriction consider that vocabulary words are binary connectors with left and right variables, called *lvar* and *rvar*.

Figure 2 illustrates schematically our basic viewpoint of the behavior of linguistic elements. A statement in the language constitutes

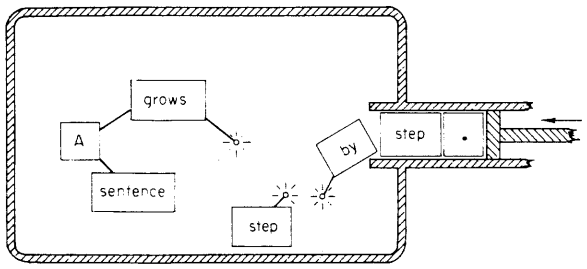


Figure 2. Reaction Vessel Analogy.

an "input string" which we may visualize as a pipe containing each of the word and symbol elements in sequence. As long as the elements are bound by the constraint of the pipe it is impossible for them to interact, but as they are pushed one by one out of the end of the pipe into a reaction vessel, "chemical reactions" of a sort take place so that elements which are attracted to each other stick together and form various appropriate sub-structures. In actual fact the reaction is a complex one depending upon the current population of the reaction vessel, and the actual sub-structures which grow are the net effect of all of the elements in the vessel. If the statement contained in the input string is meaningful, then by the time the last element is forced into the reaction vessel, one single structure, containing all of the elements of the input string, will be left in the vessel, and this structure represents the total meaning of the input string statement.

Note that except for the natural laws which govern the happenings in the reaction vessel, the entire processing is determined exclusively by the behavioral properties of the word and symbol elements themselves. Different words will behave differently; various combinations of words will cause other combinations to be formed; but even though many types of structures of many complexities may be built, the words themselves always maintain their identity and are unchanged. The analogy with the unchanging nature of chemical elements under the influence of chemical (not atomic) reactions is strikingly valid. Just as limitless chemical compounds can be made from fewer than 100 naturally occurring chemical elements, the meaningful structures which can be built from a limited vocabulary is also unbounded. It is this richness that makes language so powerful a medium.

B. *General Principles*

Using the reaction vessel analogy, it is the objective of the language theory to work out and state explicitly the behavioral laws which govern the observed operation of language. Just as in physical science there are very general natural laws akin to general principles (such as the conservation of mass-energy and the conservation of momentum), which may actively be used within the methodology of a physical theory to derive the specific formulations governing some particular area of study (for example the equations of motion of a gyroscope), the Algorithmic Theory of Language is based upon a few general principles and a methodology which yield precise formulations of algorithms for particular aspects of language. It is in fact this combination of principles, methodology, and precise formulations which correctly describe observed behavior and on occasion predict observable behavior, which makes the Algorithmic Theory of Language a bona fide theory and not merely a collection of elegant but lucky happenstances. It is solely on the basis of the soundness of these theoretical beginnings that we can look forward with some confidence to the successful achievement of a Computer-Aided Design System of the generality we seek.

In the case of the language theory, the principles are principles for algorithm construction. Recall that the basic framework within which we operate involves the explicit description of all the properties of sub-parts of problems with which we work in terms of n-component elements, and that the meanings of these components and the construction of appropriate plex structures out of the elements is to be accomplished by the execution of algorithms. Another way of saying the same thing is that the basic language of the theory itself—the way that the theory is expressed—is in terms of dynamic algorithms, rather than the static terminology of algebraic or other mathematical formulation. To work out a portion of the theory, then, consists of establishing the n-component element definitions of the parts of the problem with which we are to work, and then applying the principles as general guides to the formulation of algorithms which manipulate the components of the elements.

Although the five basic principles of the Algorithmic Theory of Language are easy to state, discussion of them would take more space than is available here so we will present as examples only the two most important principles, and use them to illustrate the beginnings of the theory. Using the terminology of *n*-component elements the *Immediacy Principle* states: "Whenever sufficient information is available for a component to be set, the setting should be performed immediately." Since multiple application of the Immediacy Principle may lead to conflicts there is also the *Stacking Principle:* "Whenever two components of the same kind require setting, the element containing the older component should be set aside on a stack (operating on the last-in-first-out principle)." These two principles find immediate application in the derivation of the Parsing Algorithm, which is the most basic algorithm of the theory, and is part of the First-Pass Algorithm.

C. *The First-Pass Algorithm*

Language consists of both form and meaning —syntax and semantics. We must develop algoriths which will transform the input string, which is the simplest kind of plex, into the First-Pass Structure, which is a more elaborate plex modelling the interlocked syntactic and semantic structure of the statement represented by the input string. The First-Pass Structure is built out of *first-pass beads.* Every vocabulary word is represented by a five-component element as follows: The *type* component tells what kind of a thing the word represents. The *ℓvar* and *rvar* components (denoted by ℓ and r respectively) point to symbols or other first-pass beads and show the left and right syntactic context of the word. The *ℓ1var* and *r1var* components (denoted by ℓ1 and r1) similarly show the left and right semantic context of the word. The pushing of words out of the pipe into the reaction vessel is accomplished by the Read-Where Routine which reads the input string one element at a time and determines where in the algorithm processing of that element should begin. We begin with the consideration of syntax alone by the derivation of the Parsing Algorithm, which sets the ℓvar and rvar components of the first-pass beads.

D. *The Parsing Algorithm*

To block out the form of the algorithm we consider first the applicability of the general principles. Since the ℓvar component is to show the left context of the word, and since the Read-Where Routine reads the input string from left to right, as soon as the word itself is encountered everything to its left will already have been considered, so that the Immediacy Principle applies and says that it must always be possible to make the final setting of the ℓvar component immediately. The rvar component, on the other hand, shows the right context, and since the right context follows the given word, the Immediacy Principle may or may not apply depending upon the particular circumstances. In the event that the right context of one word is not yet able to be determined, and another word occurs whose right context also needs to be set, the older word must be set aside on a stack, and must not be reconsidered until the right context of the more recent word has been completed, at which time the Immediacy Principle will call for its rvar to be set so that the older word can seek its right context once again. The effects of these considerations of the General Principles become clear when we consider the Parsing Algorithm itself.

Consider the input string APBQC, where *A*, *B*, *C* are symbols and *P*, *Q* are words. As the Read-Where Routine scans the input string from left to right the word *P* obtains *A* as its ℓvar and becomes the top-most thing on the stack, waiting for its rvar to be set. The Read-Where Routine continues, reading the symbol *B*, but since *B* is a symbol and has no associated first-pass bead, the Read-Where Routine will next read the word *Q*. Now the application of the Immediacy Principle says that if *B* is the proper rvar setting for *P*, that setting should be made immediately, or if *B* is the proper var setting for *Q*, then *that* setting should be made immediately. In other words the Immediacy Principle applies simultaneously to both *P* and *Q*—we say that *P* and *Q* "fight" over *B*.

As an example, consider $A + B \times C$, i.e., let *P* be the word "+", and let *Q* be the word "×". Clearly *A* is the proper left variable for +, but *B* could be either the right variable of + or the left variable of ×, i.e., + and × will fight over *B*.

We will assume for the moment the existence of an arbitrator which will decide the winner of the fight on the basis of the "chemical" attractions between the elements involved. One or the other of P and Q will have in effect a stronger attraction for B, and the arbitrator, (which is another simple algorithm derived from the General Principles but which requires more space than we have available here to describe), will decide which element wins. If Q wins, then the Stacking Principle says that Q should go on the top of the stack covering the word P, since both of them require rvar settings. On the other hand, if P wins the fight, then Q must fight with the next thing on the stack, and this fighting continues until, (by the Immediacy Principle), ultimately Q gets its ℓvar set. Then the Read-Where Routine can continue the scan of the input string.

In the example $A + B \times C$, the multiplication operator will be determined by the Fight Algorithm arbitrator to be stronger than addition, even though both "like" the variable B. Thus B will become the left variable of $\times$. Then when C and the "end-of-line" word are read in, end-of-line will loose all fights so that C becomes the right variable of $\times$, and the entire structure $B \times C$ becomes the right variable of $+$, i.e., $A + (B \times C)$. Then the entire expression $A + B * C$ will become the left variable of end-of-line, which will wait on the stack for the completion of whatever statement follows, as its right variable.

Figure 3 shows the above derivation of the Parsing Algorithm in flow-diagram language. Note that the fight arbitrator calls for help if neither the next word from the input string, (pointed to by n), nor the word on the top of the stack, (pointed to by p), is a winner. The stacking and unstacking operations are shown by trivial pointer manipulations with the s component, and the *compute type* function sets the appropriate type information into the type component of the first-pass bead which will determine its behavior in future reactions. Given the type information and information about the "likes" of the left and right sides of vocabulary words, on which the arbitrator depends, the Parsing Algorithm will make the appropriate ℓvar and r var settings to show the complete left and right context of every word in a grammatical input string.

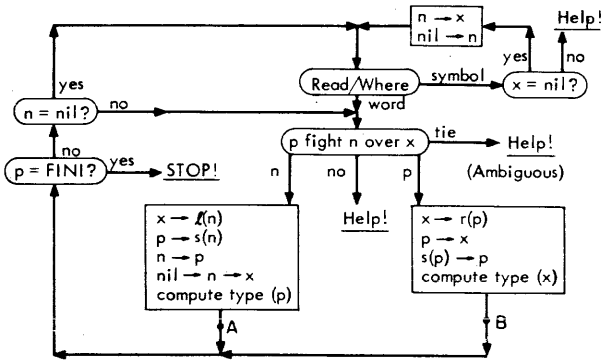


Figure 3. The Parsing Algorithm.

E. *Meaning and The Precedence String*

Once the left and right context of a word has been determined in this manner, then it is possible to consider the meaning of the word in that context. One of the novel and most powerful aspects of the new Algorithmic Theory of Language is that it explicitly includes the treatment of the semantics of the language by establishing the *precedence string* which shows the precedence or order in which words whose contexts have been determined by the Parsing Algorithm should be considered in order to build up in a step-by-step fashion the complex meaning of an over-all expression from the meanings of its subparts. Although the subject of precedence strings and semantics is very deep and only its first few stages are treated even in the definitive language theory paper, here we have space only to consider the most elementary level of *normal precedence,* to illustrate the methodology of the theory and the basic concept of precedence. In general, the precedence string shows the proper *sequence* in which operations should be performed.

Since an individual symbol represents some single physical thing or concept and has no finer internal structure for the purposes of the language theory, the meaning of a symbol is known immediately. It may be proved that for any grammatical statement, there will be at least one vocabularly word whose left and right context, as determined by the Parsing Algorithm, will consist of single atomic symbols. By the Immediacy Principle the first such atom-atom word forms the starting place for the determination of the meaning of the statement, since its meaning can be determined immediately. If that word constitutes the left or right context of another word whose other side is

atomic, then that word in turn can be evaluated, and so the process continues. For example, in $A + (B \times C)$, the product $B \times C$ can be formed immediately, and the result will then permit the + to be evaluated next.

If a word has both sides non-atomic, then after its ℓvar has been evaluated, its non-atomic rvar must again contain at least one atom-atom word where the evaluation of the rvar meaning can begin. The *minor precedence component*, ℓ1var, of the word will be set by the Precedence Algorithm to point to this atom-atom starting place of the non-atomic rvar. For example, in $(A + B) \times (C + D)$, both of the +'s are atom-atom, and after $A + B$ is evaluated, the result must be set aside while $C + D$ is formed, and only then can the × be evaluated. The minor precedence of × will point to the right hand +, and says, in effect, "make a new beginning". Every word also has a *major precedence component*, r1var, which points to the next word to be evaluated. Figure 4 shows a schematic example of a complete first-pass structure with the minor precedence components shown by dashed arrows, and the major precedence components shown by solid arrows.

To evaluate the meaning of the expression, a Precedence String Follower Algorithm will start with the first minor precedence pointer, evaluate the atom-atom word it finds there, and then follow the major precedence pointer to the next word to be evaluated. Whenever a word has a non-empty minor precedence component, the results of the evaluation made thus far are set aside on a stack, and a fresh evaluation begins at the atom-atom word indicated by the minor precedence component. Note that once the word which causes the stacking to take place because of the existence of a minor precedence component is reached *again* on the major precedence chain, then both the left and right contexts are known, so the left context is removed from the stack and used, and the processing continues.

F. *The Precedence Algorithm*

Before discussing the precedence string concept further, we will describe the derivation of the Precedence Algorithm which establishes the precedence string structure. The fact that syntax and semantics cannot be separated in a language is shown by the fact that the syntactic parsing structure and the semantic precedence

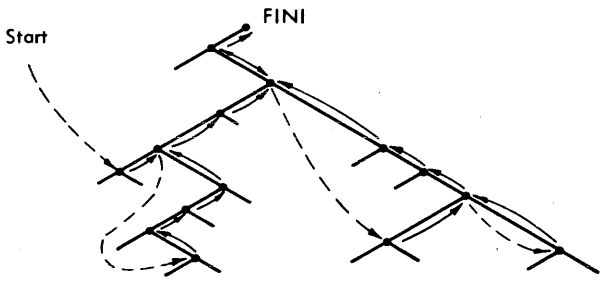


Figure 4. Example of First-Pass Structure.

structure indicated in Figure 4 are integral components of the complete first-pass structure and cannot be separated. Similarly we wish to derive a single algorithm on the basis of the General Principles which will construct the entire first-pass structure in one pass, step-by-step. Such a combined algorithm itself is a complex structure, so that we arrive at it step-by-step. The completed Parsing Algorithm is the first step, and we now will derive the Precedence Algorithm independently as a second step, and then merge the two algorithms into a single algorithm as a third step.

The Precedence Algorithm itself is trivial, as is indicated by Figure 5. We assume, for the moment, the existence of an appropriate Read-Where Routine which will decide whether a major or minor precedence component, r1 or ℓ1 respectively, is to be set by the new word, n1. After the setting has been made, we then make the new word the next word whose precedence components are to be set, pointed to by x1, and continue. Notice that in this case no stacking is required, since, as we shall see in the next paragraph, only one component can be set at a time, and no conflicts arise from the Immediacy Principle.

Examination of the first-pass structure shown in Figure 4 discloses that a *major* pre-

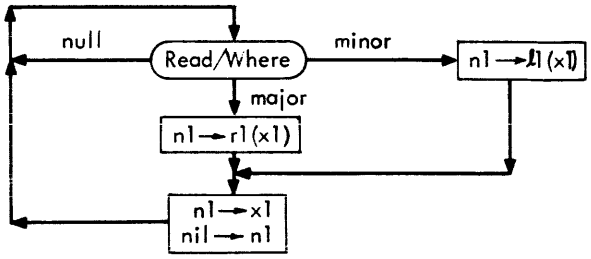


Figure 5. The Precedence Algorithm.

cedence component is to be set whenever a non-atomic ℓvar or rvar is set in the Parsing Algorithm. Similarly, considering the timing of the generation of the parsing structure by the Parsing Algorithm, a *minor* precedence component is to be set whenever atom-atom occurs, and this can in turn be interpreted to happen whenever an atomic rvar is set into a word whose ℓvar is also atomic. Thus the "input string" for the Precedence Algorithm consists of the sequence of ℓvar and rvar settings of the Parsing Algorithm, and we may think of the Precedence Algorithm as riding "piggy back" on the Parsing Algorithm.

These facts give rise to an appropriate Read-Where Routine for the Precedence Algorithm, as shown in Figure 6, which obtains its input from the labelled points in the Parsing Algorithm flow diagram shown in Figure 3. Note that the flow diagram representation for the Read-Where Routine matches exactly the verbal description of the conditions for setting of a major or minor precedence component. Merging and condensing the three separate flow diagrams results in the algorithm of Figure 7, which is the lowest level algorithm of the Algorithmic Theory of Language, since it is the simplest algorithm which results in a complete language including both syntax and semantics. Figure 8 shows the major steps in growing the complete first-pass structure for a very simple algebraic expression to illustrate the dynamics of the Parse-Precedence Algorithm in action.

III. TRANSFORMATION FROM LANGUAGE TO PLEX MODELS

As was mentioned above, the *normal* precedence that has been considered here is only the most elementary level of semantics in the Algorithmic Theory of Language. Although we do not have space to describe the complexities of the subsequent generation of still further algo-

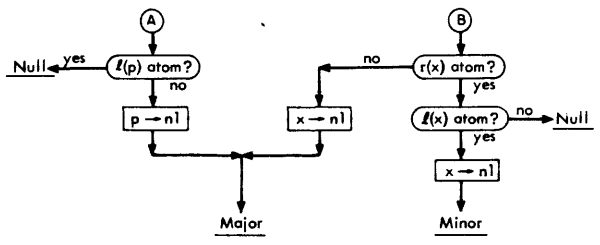


Figure 6. Read/Where for Precedence Algorithm.

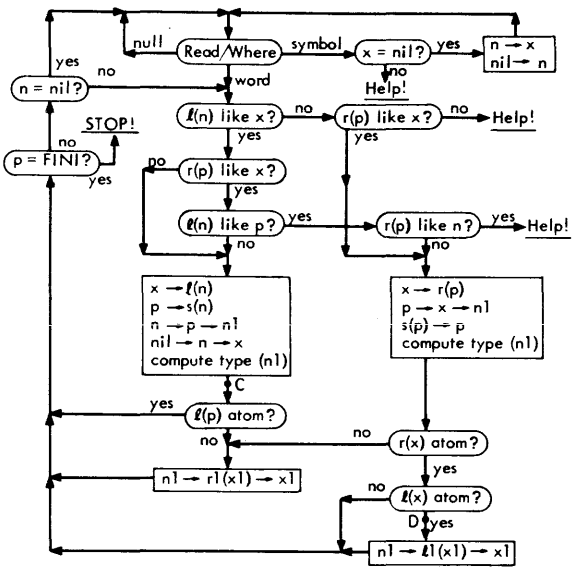


Figure 7. The Merged Parse-Precedence Algorithm.

rithms for treating more and more elaborate aspects of meaning, we close with a few remarks about the role of the precedence string and the fundamental nature of the first-pass structure.

By this time, after spending many hours describing these ideas to many individuals of differing backgrounds, the authors are well aware that there is a certain subtlety to the ideas which are being discussed here and that they lie somehow outside or on the very fringe of our communal experience, and are sometimes hard to grasp. Nonetheless, it is hoped that the following comments will serve to indicate how the complete syntactic and semantic information contained in the first-pass structure serves as the initial fundamental building block toward the establishment of the full-blown plex structures required to store all of the information about an arbitrary problem, so that the grand design of the Computer-Aided Design System can convincingly be brought into focus.

A. *The Concept of a Modelling Plex*

Recall the contention made earlier that the human's understanding of a problem being solved using the Computer-Aided System will consist of his over-all integration of the meanings, however simple or elaborate, of all of the statements made about the problem up to that point. The computer's understanding of the same problem, however, is to be contained in a

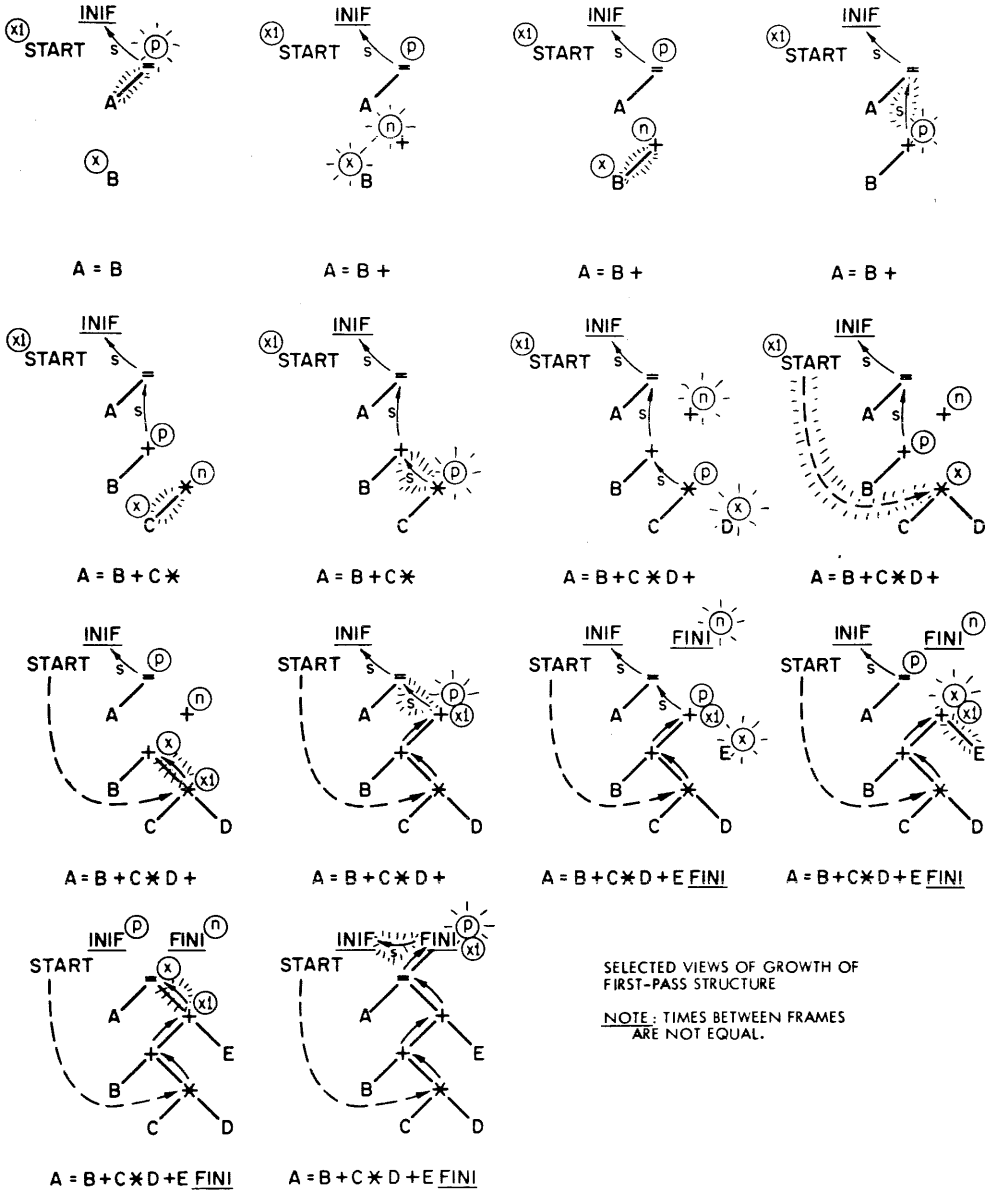


Figure 8. Stages of Growth of First-Pass Structure.

complete and detailed plex structure, which for even the simplest of problems will contain a bewildering confusion of hundreds or thousands of pointers showing all of the various kinds of interconnections between the various sub-elements of the problem. Recall also that the Algorithmic Theory of Language was introduced as the first step toward achieving this match between the human's understanding and the computer's understanding of the same problem, even though these understandings involve mechanisms which appear to be so different—the completely unknown physiology and psychology of the human mind and the incomprehensible plex structure. The way in which this vital linkage of the two understandings is to take place is the subject of this discussion.

The crucial point to be made is that there is a sharp and very important distinction between statements *about* a problem and the problem *itself*. The situation is somewhat reminiscent of that branch of classical philosophy which questioned the existence of a reality without an intellect to be aware of that reality. In fact, the

singular noun *reality* may serve our purposes better than the word *problem*. Thus there is a "distinction between statements *about* a reality, and the reality *itself*". Problemness itself is merely another reality superimposed on some other, supposedly more *real* reality. The thing which must be perceived clearly is that however many things may be *said* about something, even though those sayings may specify all aspects of that something, they still remain merely sayings.

In order to achieve our broad Computer-Aided Design objectives, we must create within the computer a precise representation of the actual reality itself. This is why the concept of modelling has been so emphasized. In fact, it may properly be said that the modelling plex is *the*, and the *only*, reality regarding any problem being solved. The match between this idealized abstract reality and the *real* reality depends entirely upon the success of the intellect generating the sayings in perceiving and giving expression to all of the pertinent aspects of that reality.

B. *Operators for Plex Transformations.*

So we address again the question of how the language statements become the modelling plex. The algorithms of the Algorithmic Theory of Language provide the answer with regard to the statements themselves. The first-pass structure for a statement *is* the proper modelling plex for the statement, showing explicitly the syntax and semantics of the statement. The next step is to transform the meaning of a statement into appropriate settings of the components of elements in a modelling plex. This important transformation is achieved by *operators* following the precedence string.

In effect, operators are the elements of "intellect" for the computer, for they generate internal "understanding" in the form of modelling plexes, which constitute the memory system. There will be many operators which perform many functions, but all operators follow the same basic principle—they transform the successive meanings encountered while following a precedence string into some appropriate other form, and thereby transform one plex into another plex. Sometimes the plexes are modelling plexes and sometimes they are first-pass structures, i.e., sometimes operators are concerned with problems and sometimes with descriptions. The particular ways in which operators are defined and the ways in which they are used determines the over-all result, so that this paper presents only the general idea of operators and is in no way a definitive treatment of the subject.

The basic idea of an operator may be illustrated by a simple example. Figure 9 shows the first-pass structure for the statement "Line A connects point B and point C." Consider an operator which is appropriate for topological consideration of points and lines. It might function as follows: The operator begins at START in Figure 9, and when the word *line* is

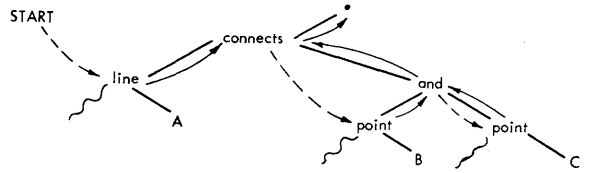


Figure 9. Line Description.

encountered on the precedence string, an empty 4-component element is obtained from free storage in the computer, the code for "line" is placed in the type component, and the code for "A" is placed in the name component. The operator then follows the precedence string to the word *connects*, but finding a non-empty minor precedence component, it proceeds to the word *point*. There it obtains an empty 4-component element and places "point" and "B" in the proper components. Next the word *and* sends the operator on to do the same for the other word, *point*, creating an element for "point" "C". Now the word *and* is encountered for the last time, and it causes the operator to associate the two "point" elements, either by setting up a "pair" type of element with pointers to the "point" elements or by some other mechanism. Finally, the word *connects* is reached again and it causes the remaining components to be set, as shown in Figure 10. The *period* terminates the operator.

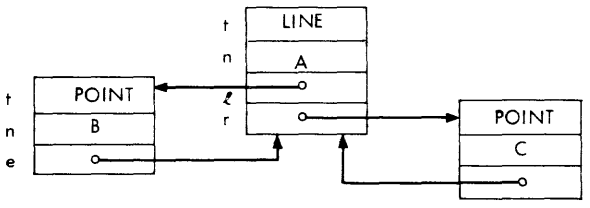


Figure 10. Line Model.

Although such an operator might be constructed quite differently in practice, the example does show how the first-pass plex of Figure 9 is transformed into the modelling plex of Figure 10. Clearly, analogous and much more sophisticated and elaborate operators are possible, and the technique is completely general. Some such operators are described in the following sections, but it should be emphasized that the intent of this paper is to explain the conceptual and theoretical framework for an evolutionary Computer-Aided Design System.

There exist many techniques which can merge and eliminate theoretical steps in some cases, so that working systems may not exhibit the fine structure of the theory. An example of this is shown in the following papers on graphical language facilities,[3, 4] which have been written thus far with standard coding techniques which skip over many of the theoretical stages referenced in this paper. Nonetheless in these special cases the theoretical framework may be considered to be present, and in order to permit the further evolution of even more elaborate facilities in a natural way without specifically working out each detail, we need the theoretical understanding, and in fact must attempt to build in the theory into the internal workings of the system itself. Since our long range objective is not merely to provide the user with those pieces of a system which we ourselves are able to conceive of and construct, but instead is to provide a system which is adaptable to each user even though he may not be a sophisticated computer programmer, the unifying theoretical framework is not only of interest in and of itself, but is essential to the proper execution of our mandate.

The simplest form which an operator can take is that of a Markov Normal Algorithm[5] or LISP conditional expression,[6] i.e., in general an operator is a recursive function consisting of a list of rules each of which has a left and a right side. For each rule, the left-hand side asks questions about some of the arguments of the operator, and the right-hand side describes certain operations to be performed on some of the arguments of the operator. To evaluate the operator we start at the top with the first rule, and if, substituting arguments, the left-hand side of the rule is *true*, then we execute the right-hand side, and otherwise continue on to the next rule. Recursiveness occurs whenever the right-hand side of a rule references the rule itself, and in this case there must exist somewhere in the operator a right-hand side containing a termination signal or no recursive reference. In general an operator may have a more complex structure in which all left sides are evaluated simultaneously rather than sequentially,[7] but the slightly different mathematical entity which this form of operator represents has not been studied theoretically yet, and we will not describe it here. In many cases it is not necessary to have recursive operators, since they will be used following a precedence string which presents information in precisely the correct order, so that recursive operations are not necessary, but if a proper precedence string is not available, then, in general, recursion is required.

Operators may obtain arguments from one of three sources: from a modelling plex or from a first-pass structure, (i.e., from a model of a problem or a model of a statement about a problem), or from another operator. The right side of a rule may be any algorithm, and thus we may have operators for any purpose.

C. *Operator Examples*

To clarify the concept of an operator and how it works, we consider two of the simplest operators: the "precedence follower" operator and the "calculus derivative" operator. The precedence follower operator is used to control the routing of another operator through a first-pass structure. In other words, other operators normally ride piggy back on the precedence follower operator if they are applied to a first-pass structure. The precedence follower operator takes a first-pass bead as argument and follows the minor precedence component if one exists, and otherwise follows the major precedence component. The result of execution of the operator is a pointer to the next first-pass bead which will be used as the argument of some other operator. The rules of the precedence follower operator may be written as follows:

$\text{prec}(x)$: $\ell 1(x) \neq \text{nil}$
and x is new $\Rightarrow \ell 1(x)$, Terminate
otherwise $\Rightarrow r1(x)$, Terminate

I.e., if the minor precedence component, $\ell 1$, of the argument, x, is not empty and x has not been encountered before, the value of the operator prec(x) is the contents of that component, (i.e., a pointer to the "next" thing on the precedence string). Otherwise the value of prec(x) is the contents of the major precedence component (which again is the "next" thing on the precedence string). The precedence string follower may be used to carry another operator, say *doer*(y), along the precedence string by writing

doer (prec (START))

where START contains a pointer to the beginning of the precedence string in a first-pass structure, such as Figure 9. Then whenever *prec* terminates, *doer* will perform its function on the value of *prec*, (i.e., the thing pointed to as "next" on the precedence string), and then when *doer* is done doing what it is supposed to, *prec* will be reactivated to take another step along the precedence string.

The output of the precedence follower is a pointer to a first-pass bead, and to simplify discussion we consider operators which have first-pass beads as arguments as a special case with a built-in "matching operator" which compares the first-pass structure whose top is the specified argument bead with the left side of a rule, and the matching operator is *true* if, with appropriate substitutions, the structures are the same. Using this convention we may consider the special symbol manipulation operator which processes the first-pass structure of an algebraic statement into the first-pass structure of the calculus derivative of that expression.

The algebraic derivative operator is defined by the rules:

Rule I $d(u + v) \Rightarrow dx + dv$

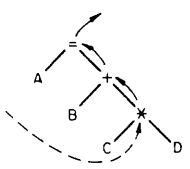
Rule II $d(u \times v) \Rightarrow u \times dv + v \times du$

Rule III $d(u = v) \Rightarrow du = dv$

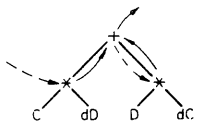
The simple algebraic expression

$$A = B + C \times D$$

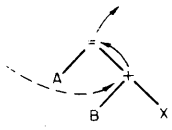
is processed into the First-Pass Structure



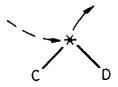
Starting along the precedence string the structure $C \times D$ is matched by Rule II, the right-hand side of which generates $d(C \times D)$



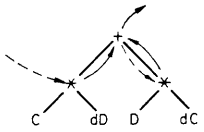
Note that after this processing the remainder First-Pass Structure can be considered to be



where X is

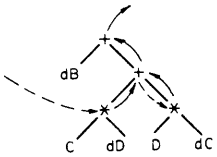


and dX is

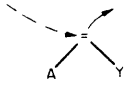


Next along the precedence string is the word + and it is matched by Rule I provided that the right argument of +, i.e. $(C \times D)$, is substituted for v. Inasmuch as the precedence string assures that whatever arguments + has, they have already been processed, the rule is still operating on an atomic level with respect to itself and it can be fully applied.

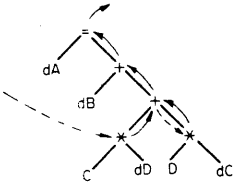
The structure generated at this step is



The unprocessed First-Pass Structure can be viewed as



where Y is $B + C \times D$ and dY is the structure generated above. Finally by application of Rule III there results



which corresponds to

$$dA = dB + C \times dD + D \times dC$$

IV. THE DESIGN SYSTEM IN USE

The derivative operator is a *description* operator since its output is a first-pass structure, i.e., the operator describes its input by making a meaningful statement about it. We now use this concept to outline how an actual design problem would be solved using the Computer-Aided Design System by making statements which trigger complex interactions of description and construction operators. Computation functions or subroutines may of course be viewed as computation operators, and we assume that where appropriate they would be triggered as well.

A. *General Description*

The designer begins conversing with the system by describing whatever features of his problem are uppermost in his mind. The language used for this description process may be verbal, using a typewriter keyboard or pushbuttons and control knobs, or graphical, using a light pen-oscilloscope combination. Since the light pen is used demonstratively and in conjunction with pushbuttons, keyboard statements, and internal program states, each separate action with a pen (even though blurred together by the smoothness of motion and by the speed of the computer) is a separate word with "likes" giving the effect of chemical attractions, so that the result is the growth of an appropriate first-pass structure representing whatever the designer says.

At appropriate times, perhaps at well-defined ends of statements or perhaps on-the-fly, some of the things which have been said will call for action by the computer other than just execution of the First-Pass Algorithm and generation of further first-pass structure. At these times suitable operators are "turned loose" on the precedence string of the existing first-pass structure and perform their function as outlined above. If an operator is a description operator, its effect will merely be to elaborate on the meaning of what has explicitly been said. Many operators will be construction operators, similar to the one described earlier for constructing the plex for a line with its end points, which go to free storage if necessary to set up connectors in a modelling plex which represents the computer's growing understanding of the problem. Some executable statements will turn loose description operators on the modelling plex itself, and these operators will follow pathways of their own choosing, like a mouse solving a maze, chattering away and generating descriptions of what they find. These descriptions are in turn processed by the First-Pass Algorithm and the meaning may be processed further by operators which elaborate the meaning, or construct further modelling plexes, or modify existing plexes. The chain of effects triggered by a simple action by the designer may be astronomical in a well-developed Computer-Aided Design System.

But the common framework is there. The entire process, however elaborate, consists of meaningful linguistic descriptions being transformed into other descriptions or models. The descriptions themselves come either from the man or from interpretations of various modelling plexes by appropriate description operators. The triggering of operators which transform meanings, construct plexes, or describe plexes to further operators or to the outside for human consumption is all automatic, and is similar to the bubbling activity which one observes in the living cell. Since the entire process is based ultimately upon the interactions between the *meanings* of the many elements involved, and since the sorting out of what things go together and what things do not go together is handled automatically by the "natural laws" of behavior which are built in, the designer on the outside has no conception of the chaotic activity inside the system, but sees only external effects appropriate to his mode of understanding.

There are many further elaborations on this basic concept of how the Computer-Aided Design System should operate, and many things which could be said about the importance of such a proper, theoretically sound internal construction of the system to the prognosis for the evolution of a full-scale system meeting our mandate, but these comments will be reserved for future papers as the system itself takes concrete shape. Instead we close with an example drawn from the field of servomechanism design to show some of these concepts in action.

B. *Servomechanism Design Example*

A typical problem in servomechanism design is as follows:

Given the fixed elements of a linear feedback control loop, design an appropriate compensation network to meet given steady state and dynamic specifications. The problem is to be solved using frequency response techniques.

A brief review of the required techniques appears in Appendix A, together with definitions of the terminology used. The design method is a trial and error procedure with the following steps:

1) Compute the frequency response of the uncompensated open loop.
2) If not satisfactory select a compensation that will hopefully introduce the desired changes.
3) Compute a new frequency response of the compensated open loop to show the actual effect.
4) If this is also not satisfactory, repeat steps 2) and 3) until the specifications are met.

To solve this problem using the Computer-Aided Design System the following operators are needed:

1) Matching, substituting and selecting operators
2) Signal flow interpreter
3) Transfer function operator
4) Complex number simplification operator
5) Complex number magnitude and phase operators.

The functions and definitions of these operators appear in Appendix B and may be defined for the system at any time before usage. Also operators may be defined as the composition of several other operators, so that for the present problem we may define a *frequency response* operator as the composition of the complex number simplification, magnitude, and phase operators. With the necessary operators defined, we may proceed to the solution of the problem.

The first step is to describe the topology of the feedback loop. This is best accomplished by means of graphical input using n-component elements for the various picture pieces with *type*, T, *name*, N, *input from*, I, and *output to*, O, components.

By an appropriate sequence of light pen motions the picture of Figure 11 can be drawn and processed to produce a plex structure which we call the modelling plex for the problem. Notice that the act of drawing is a linguistic process with its own grammar rules, i.e., boxes can be connected only by connectors, *input from* points can be joined only to *output to* points, etc. The resulting modelling plex for the feedback loop is shown in Figure 12.

The modelling plex is a representation of the topology of the loop and as such it can be interpreted in many different ways. In particular we are interested in the signal flow through the loop. On command, the Signal Flow Interpreter

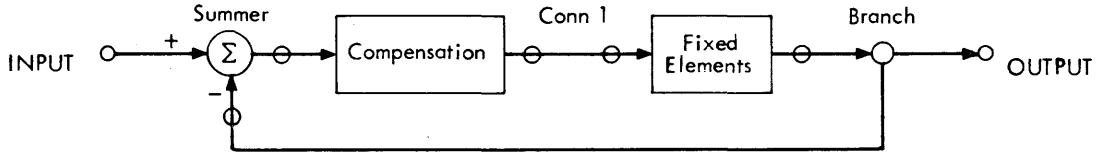


Figure 11. A Typical Control Loop.

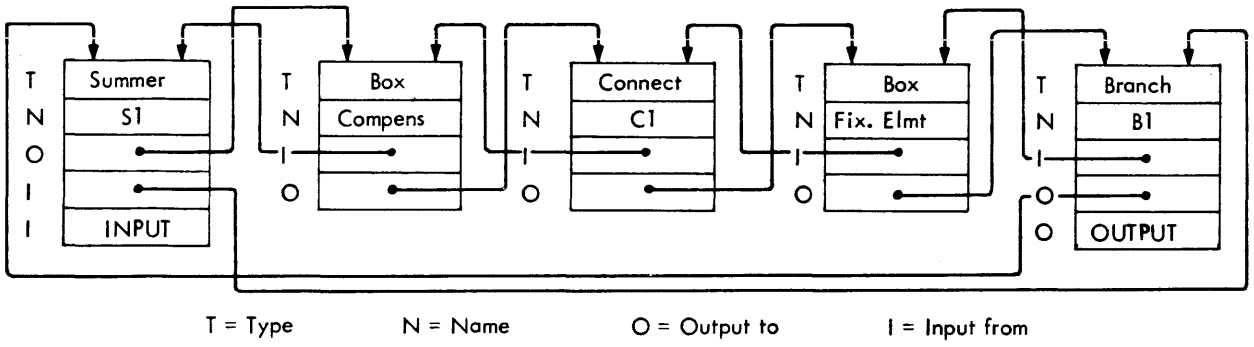


Figure 12. The Modelling Plex for the Control Loop.

Operator runs through the modelling plex, following the same path an actual signal would take, and producing a language description of the meaningful components. The resulting language description is processed by the First-Pass Algorithm to generate the First-Pass Structure which models the description of the problem in terms of signal flow, as shown in Figure 13.

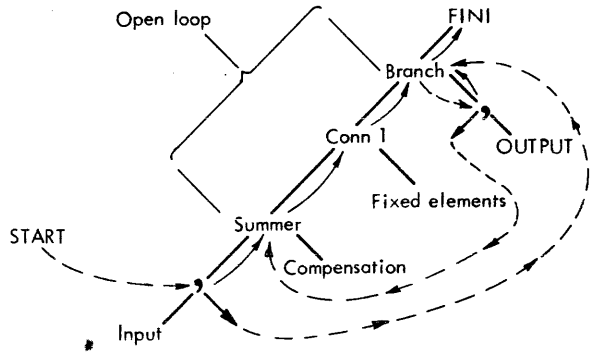


Figure 13. First-Pass Structure for the Loop.

Using the Selection Operator the designer may select any substructure that he wishes to process further. These substructures may be selected by pointing at them in the current problem display, or by assigned names suitable for typewriter reference. For this problem we choose to isolate the *open loop*, defined as the compensation and fixed elements of the servomechanism.

To obtain the open loop transfer function, a command is issued to apply the Transfer Function Operator to the open loop description, substituting the mathematical multiplication operation for topological connectors.

Then the designer may define the fixed element and the compensation by giving their transfer functions by typing:

The fixed element IS 1/(s ×(s + 1));
The compensation IS 1

which causes this information to be introduced into the open loop description. The transfer function as we have it now is a detailed description of one aspect of the problem. It has been obtained by successive transformation of the meaning of several modelling and description plexes.

In order to obtain the frequency response of the open loop, further transformations of meaning are required. The designer can now command the system to apply the Frequency Response Operator to the open loop structure. This operator is the composition of four other operators applied successively. First $j\omega$ is substituted for s, then the Complex Number Simplification Operator sets things up so that the Magnitude and Phase Operators can meaningfully be applied, and the resulting description is used to compile a calculation program. Figure 14 shows the description plex after the Simplification Operator has been applied, and Figure 15 shows the result of applying the Magnitude and Phase Operators to the plex of Figure 14.

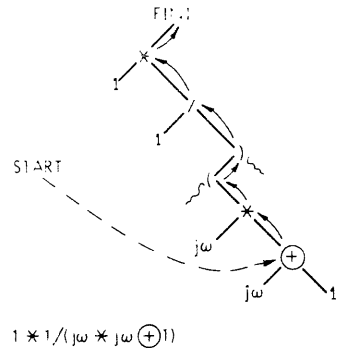


Figure 14. The Open Loop Plex after Simplification.

The result of the Frequency Response command is a program to compute numerical values of the magnitude and phase of the open-loop transfer function. These numerical values can conveniently be displayed to suit the designer, who can then decide what sort of compensation network he would like to use. Since the structure of the problem has been preserved, changes are very easily introduced at any level. In particular to change the compensation at the transfer function level the designer may type

The compensation IS (5 × s + 1)/(50 × s + 1)

A new frequency response can now be obtained in the same way as before and so the design process continues.

B. *Concluding Remarks*

In addition to the operators outlined here, operators have been devised for simplifying algebraic expressions and boolean expressions, performing useful tree structure manipulations, optimizing the evaluation of decision functions based on probabilities or costs or both, perform-

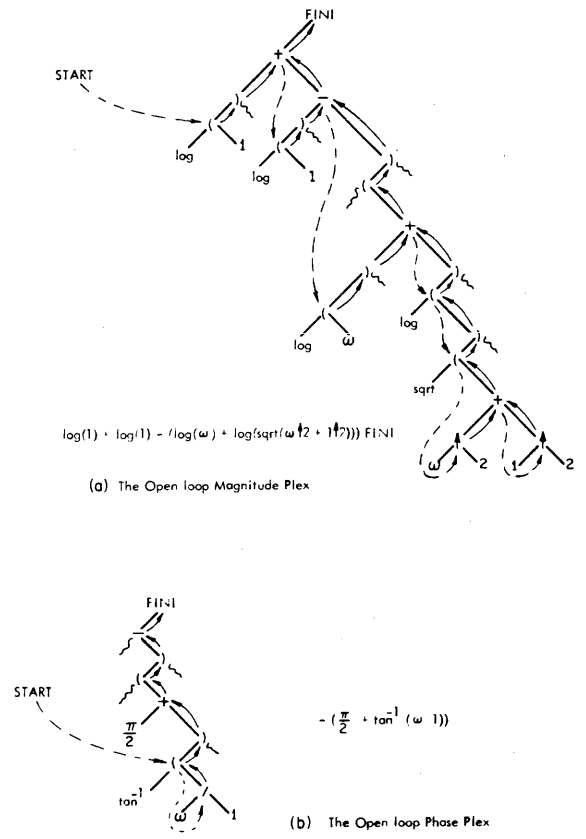


Figure 15. The Magnitude and Phase Functions.

ing simple natural language translation, and carrying out many functions which arise in the efficient compilation and execution of efficient computer programs. Many other operators useful in various design problems can also be envisioned, but this area has hardly been tapped as yet.

The future developments which can be sensed as growing out of the elaboration of the operator concept indicate that the algorithms and methodology which will result might appropriately be termed an Algorithmic Theory of Meaning. In these terms it is difficult to see the dividing line between the Algorithmic Theory of Language and the Algorithmic Theory of Meaning, since the features of both become interlocked and intertwined as more elaborate and richer modes of operation are considered.

All of the things which have been discussed here are presently being reduced to practice, and the implementation of the beginning stages of a full Computer-Aided Design System is well under way. In addition to the language theory paper which has been referenced, a series of papers will shortly be forthcoming which will elaborate on the Algorithmic Theory of Language, provide details of Operator Theory, and describe completely the compiler and programming system which embodies all of these theoretical features and forms the nucleus for the full system.

Whatever the further development, it is quite apparent that although it would be foolish to claim that the full scope of the mandate for a completely general Computer-Aided Design System can be met directly using the techiques presented here, nonetheless it is impossible to put bounds or limitations on the potential capabilities of the system taking shape. It is hoped that this brief description of the first several steps along the way to a powerful Computer-Aided Design System will enable others to share our confidence and enthusiasm for these beginnings, so that they may join in the fascinating pursuit of a generalized scheme for problem solving.

APPENDIX A—SERVO TERMINOLOGY

Transfer Function

The transfer function of an element is defined as the Laplace Transform of the impulse response of the element

$$H(s) = \int_0^\infty h(t)e^{-st}dt$$

where: $h(t)$ is the impulse response
t is time
s is the complex variable $\sigma + j\omega$

The transfer function relates the input and output of an element as follows:

Output = Transfer Function × Input.

Frequency Response

The frequency response is defined as the magnitude and phase of the transfer function as a function of ω when $s = j\omega$.

Physically a point in the frequency response curve means the following:

When the input to the system is a sine wave, the *magnitude* is the logarithm of the ratio of the amplitude of the output to the amplitude of the input, while the *phase* corresponds to the phase difference between the output and the input.

APPENDIX B—OPERATOR DEFINITIONS

Operators Used in The Servo Design Example

Only those rules which are needed and cause a change are given

1. *Transfer function operator*
   Argument: A First-Pass Structure
   Operation:

   $$u \text{ connector } v \Rightarrow \text{substitute } (u \times v)$$

2. *Complex number operators*
   x and y are real quantities
   c is a complex quantity
   $\oplus$ is the complex number symbol
   $j = \sqrt{-1}$
   a) Simplification operator
   $jx + y \Rightarrow$ substitute $(jx \oplus y)$
   b) Magnitude operator
   $jx \times c \Rightarrow$ substitute $(\log(x) + |c|)$
   $jx \oplus y \Rightarrow$ substitute $(\log(\text{sqrt}(x \uparrow 2 + y \uparrow 2)))$
   $x/c \Rightarrow$ substitute $(\log(x) - |c|)$
   $x \times c \Rightarrow$ substitute $(\log(x) + |c|)$
   c) Phase operator
   $jx \times c \Rightarrow$ substitute $\left(\frac{\pi}{2} + \text{phase } (c)\right)$
   $jx \oplus y \Rightarrow$ substitute $(\tan^{-1}(x/y))$
   $x/c \Rightarrow$ substitute $(-\text{phase } (c))$
   $x \times c \Rightarrow$ substitute $(\text{phase } (c))$

BIBLIOGRAPHY

1. ROSS, D. T., "A Generalized Technique for Symbol Manipulation and Numerical Calculation," *Communications of the Association for Computing Machinery*, Vol. 4, No. 3, pp. 147-150, March 1961.
2. ROSS, D. T., "An Algorithmic Theory of Language," Report ESL-TM-156, Electronic Systems Laboratory, Massachusetts Institute of Technology, 67 pp. November 1962. To be published in the *Journal of the Association for Computing Machinery* in 1963.
3. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Communication System," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963. (This Volume).
4. JOHNSON, T. E., "Sketchpad III, A Computer Program for Drawing in Three Dimensions," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963. (This Volume).
5. MARKOV, A. A., *Theory of Algorithms*, Academy of Sciences, USSR, 1954. (Translated by Jacques T. Schorr-Kon, et al., Office of Technical Services, Washington), 444 pp.
6. MCCARTHY, J. "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1," *Communications of the Association for Computing Machinery*, Vol. 3, No. 4, pp. 184-195, April 1960.
7. ROSS, D. T., "Research on the Evaluation of Simultaneous Logical Functions," *Investigations in Computer-Aided Design, Interim Report No. 1*, Report 8436-IR-1, Electronic Systems Laboratory, Massachusetts Institute of Technology, pp. 47-66, May 30, 1960.

MAN-MACHINE CONSOLE FACILITIES FOR COMPUTER-AIDED DESIGN

Robert Stotz
Electronic Systems Laboratory
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

EQUIPMENT

The backbone of the man-machine communication link in Computer-Aided Design is a console whose principal components are the display scope and the light pen. The display scope is an ordinary cathode ray tube which is controlled by the computer by means of program instructions. It allows the computer to output to the man rapidly in easily interpreted graphical form. The data displayed can be textual, pictorial, or a combination of the two. The light pen is a photosensitive device which responds to the light generated by an intensified point on the scope face and which amplifies, shapes and transmits this response back to the computer where it can be tested by the program and used as a branch condition. The display scope and light pen form, so to speak, the paper and pencil of the designer, but they possess some extremely useful additional properties which open a whole new expressive medium. The following papers describe some of the fascinating and invaluable facilities which are provided by this basically simple hardware. In this paper we present some basic information about the hardware itself and describe some of the sophistications appropriate to the Computer-Aided Design problem.

To provide flexibility of input, particularly regarding the control of the displays and the interpretation of light pen actions, the console should also contain various knobs, switches, and buttons to complement the display while manipulating these controls. In addition, a typewriter is incorporated to permit input of textual and numeric information and hard copy output.

The Light Pen

The light pen designed at Lincoln Laboratory is a hand-held cylinder with a photocell mounted inside at one end and a wire leading back to the computer at the other. In addition to the photocell, a one transistor preamplifier is also housed inside the tubular body of the pen. The photocell circuit is designed to respond to only the initial flash of the phosphor and is insensitive to the persistence observed by the human. The photocell signal is a.c. coupled onto the d.c. voltage line back to the amplifier section, so that only a single coaxial wire is required to carry the power to the pen and the signal back to the computer. The light pen housing designed and used by the Electronic Systems Laboratory has a variable focus lens mounted in front of the photocell. This makes the pen more sensi-

This work has been made possible through the support extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory by the Manufacturing Technology Laboratory, ASD, Wright-Patterson Air Force Base under Contract No. AF-33(600)-42859. It is published for technical information only and does not necessarily represent the recommendations or conclusions of the sponsoring agency.

tive and yields a variable field of view depending on the focus setting. The field of view of this pen ranges from ⅔ inch diameter down to 1/16 inch diameter and for the larger settings is almost cylindrical so that scope-to-pen distance is not important. Pens made from fiber optics material with photomultipliers at their base have recently been introduced. Their rapid response, light weight and high sensitivity make them very promising for future work.

The preamplifier signal is boosted by a solid-state amplifier which drives a flip flop. The computer then samples the state of this device to determine if the point displayed was "seen."

The light pen can be used to simulate the "drawing" ability of a real pen by having the computer program "track" its arbitrary motion over the scope face. The principal of the tracking program is to detect the edge of the field-of-view of the pen at several divergent points and compute the center of gravity of these, which is considered the pen position. Figure 1 illustrates one of the simplest of the many schemes for establishing these edge points. The last previous center point is used as the starting point for four radially drawn lines which are each individually terminated when no longer "seen" by the pen. The center of gravity of these four end points is the new pen location and is used as the center position for the next iteration. This process is continually repeated, each iteration requiring from 1 to 3 milliseconds. This is fast enough to allow normal writing speed without "losing" the program.

The light pen also has the unique power of being able to identify points or lines to the computer. When used in conjunction with special buttons or knobs or keyboard input these "identified" points or lines can be assessed in any number of ways. Solitary points can be interpreted as special test points by the program, so that when these spots are "seen" by the pen,

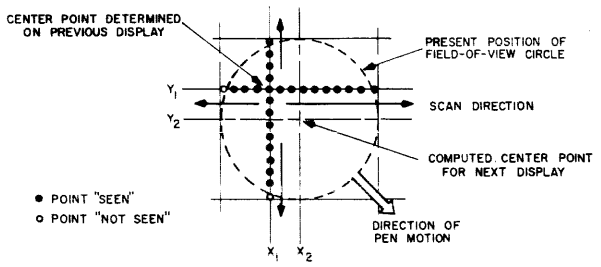


Figure 1. Basic Light Pen Tracking Pattern.

the program branches. Such points, called Light Buttons, can be used just as toggle switches or they can be arbitrarily placed on the scope face, even interspersed within the display picture itself.

Display Scope

The most common form of the display scope today is one which intensifies a single spot at a time. The position of a point is specified by a computer display instruction giving horizontal and vertical coordinates. A typical display scope will have $2^{10} = 1024$ unique horizontal coordinate values and a similar number of vertical values for a total of over 1,000,000 discrete points available. These are usually distributed over an active surface of about 10 inches x 10 inches. Display times range from 3 to 140 microseconds per point depending on the scope size and type of deflection circuitry. Pictures are drawn by outputting a list of points by repeated display instructions. This list is called the "Display File."

The next generation of scopes have line and character generation. There are a number of such units commercially available today, and they vary in emphasis depending on their particular application. Most of the character generating types come with keyboard inputs and are used as flexible typewriters. They often feature a light pen or cross hairs to identify individual characters or points. Line segments are either formed by the character generator as special characters or are produced by a line generator which is an additional piece of hardware. Depending on the particular line generator, lines are specified by their incremental components, by their slope and length, or by their end points. In nearly all cases the starting point is the end point from the last display instruction.

The advantages of these scopes are two-fold. More data can be displayed using less computer storage and less computer generation time. Higher plotting speeds evolve because a character or vector can generally be drawn nearly as rapidly as a single dot in the ordinary point-for-point scope. Computer time is saved because it no longer must produce the list of points which constitute the line. The display file is considerably reduced, too, since it now is just a list of vectors instead of an extended

file of single points. Almost all sophisticated man-machine consoles have adopted the character generator and some form of line generation.

Another popular feature of display scope systems is local buffer storage for the display file. This reduces the main frame load immensely since the regeneration of the display is completely handled by the buffer system. Only when the picture is to be altered is the main frame required to process the display. In most applications this operation occupies a small percentage of the time. The next logical extension is to provide logic associated with the memory to provide certain control facilities in the remote scope itself. These control functions can be expanded and extended until the buffer unit becomes a general purpose computer in its own right.

DISPLAY REQUIREMENTS FOR COMPUTER-AIDED DESIGN

At M.I.T. the requirements for Computer-Aided Design have led to investigation of a different type of display sophistication. Many studies of the Computer-Aided Design Group are concerned with three-dimensional objects with curved surfaces. Therefore, a display scope capable of generating three-dimensional curvilinear figures with convenient control of translation and rotation would be extremely useful. The approach that has been taken is to provide a function generator capable of generating straight lines or second-order curves, and to have special purpose computing equipment in the display unit which can perform the three-dimensions to two-dimensional axonometric projection computations. This frees the main frame from the laborious task of recomputing the entire display each time the picture is changed in any way. If the picture is to be rotated, translated or magnified the computer merely has to alter a few key registers in the display unit and then output the display list as before.

The axonometric projection was chosen because of its simplicity and its compatability with most existing engineering drawings. More complex projections such as perspective or even stereoscopic views are feasible with added equipment, but at this juncture the need for these more intricate views is not established. It may for instance be possible to produce a completely satisfactory three-dimensional effect by modulating the beam intensity with a signal which is proportional to the depth of the point displayed. Until more experience is gained in the manipulation of three-dimensional objects, the added facility does not warrant the expense. In the meantime these sophisticated display projections can be generated by the computer and output to the scope as a two-dimensional picture, which the three-dimensional system is completely capable of handling. If after further study it is decided to include the equipment to accomplish these more general transformations, the hardware can be easily added as a modular package.

M.I.T. DISPLAY SYSTEM

A system is presently being designed for the Electronic Systems Laboratory which will display on a scope a standard two-dimensional plot or a rotated axonometric projection of a three-dimensional line drawing. The system will be capable of generating first- and second-order lines and will provide easy means for the computer to translate or rotate the picture, or to magnify or demagnify it. Techniques are incorporated to prevent plotting when the limits of the scope edge are reached.

Figure 2 illustrates this system. It is made up of three main parts. The first part is the line generating unit, the second contains rotation matrix multipliers and the third, two accumulating registers which hold the horizontal and vertical coordinate information for the scope deflection amplifiers.

The line generator inputs are the three-dimensional parameters of the line to be drawn given in terms of a fixed cartesian coordinate space x, y, z. Its outputs are three time-varying signals representing the x, y, and z components of the input line. This line generator is made of three 10 bit Incremental Integrators (such as a Digital Differential Analyzer) one for each component. The time varying signals thus produced are in the form of pulse trains. By establishing gated feedback links between two of the DDA's the line generator is made capable of producing a straight line, circle, hyperbola or parabola, selectable by computer program.

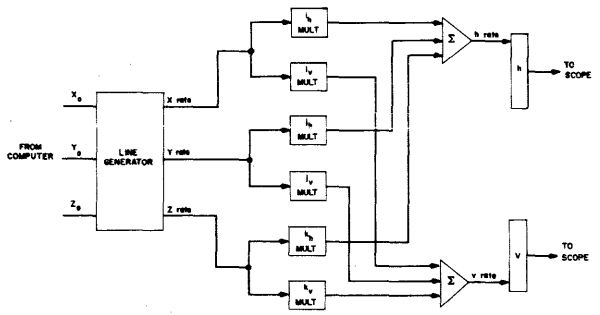


Figure 2. Block Diagram for Display System with Vector Generation and Rotational Capabilities.

The rotation matrix essentially applies a matrix multiplication to the x, y, and z pulse trains to provide proper time-varying signals for a display of the same figure in a rotated coordinate system. The amount of rotation provided is preset by the computer in storage within the rotation matrix unit. Since the x, y, z outputs of the line generator are in the form of pulse trains, the matrix multiplication can be conveniently accomplished with Binary Rate Multipliers, which multiply a pulse rate by a binary fractional value to generate a reduced rate. The outputs of the three BRM's containing horizontal components of the rotation matrix are summed into the h accumulating register while vertical components feed the v accumulating register.

The accumulating registers are a form of summing integrator. Their instantaneous value represents the correct horizontal and vertical position of the electron beam. Thus they can be used to drive directly the digital-to-analog converter leading to the scope deflection amplifiers.

The presence of a rotation matrix in the system which can be arbitrarily loaded by the main frame computer eases the requirements on the line generator considerably. The Rotation Matrix provides the facility for a linear transformation of the line generator output. By using this transformation any generalized ellipse can be formed from a circle. Similarly any parabola can be obtained from a single parabolic form, and any hyperbola can be produced from a single hyperbolic form. Thus the line generator is required to generate just one form of the ellipse (the circle), the parabola and the hyperbola. A single pair of DDA's will accomplish this.

To generate a rotated straight line in this machine the input vector A is specified by its x, y and z components (A_x, A_y, A_z). The components refer to the dimensions of the line in the fixed coordinate space. These components each have components of their own in the coordinate space of the scope (h, v, d; horizontal, vertical and depth respectively). The multipliers i_h, j_h, and k_h represent the horizontal components of unit vectors in the x, y, and z directions. Therefore A in scope coordinates is

$$\begin{bmatrix} A_h \\ A_v \\ A_d \end{bmatrix} = \begin{bmatrix} A_x \cdot i_h + A_y \cdot j_h + A_z \cdot k_h \\ A_x \cdot i_v + A_y \cdot j_v + A_z \cdot k_v \\ A_x \cdot i_d + A_y \cdot j_d + A_z \cdot k_d \end{bmatrix}$$

Since only the horizontal and vertical components appear on the scope, computation of the A_d component is not mechanized. More sophisticated projections would require this depth computation.

It can be seen that the computer controls the setting of several different registers within the display system. Besides the line-drawing data, it must supply the rotation numbers, the initial setting of the h and v accumulating registers and a magnification control register setting which allows increase of the picture size by powers of two by controlling the scaling of the input values of x, y, and z (in between the scales may be obtained through the rotation numbers). Furthermore, the line generator itself can have several different modes of operation. To provide the display unit with a means of identifying the type of data being presented, each word contains a control field. The bits of this field are decoded by logic in the display unit to set up appropriate gating.

The normal display list then consists of a heading, which loads the starting point and magnification control and the values for the rotation matrix components, followed by the body containing the line information. Translation and rotation of the entire picture is accomplished by the program merely altering the elements of the heading.

SIMULATION

To investigate the quality of the figures produced by the display system a computer program was written to simulate the entire machine. Provisions were built into it to test

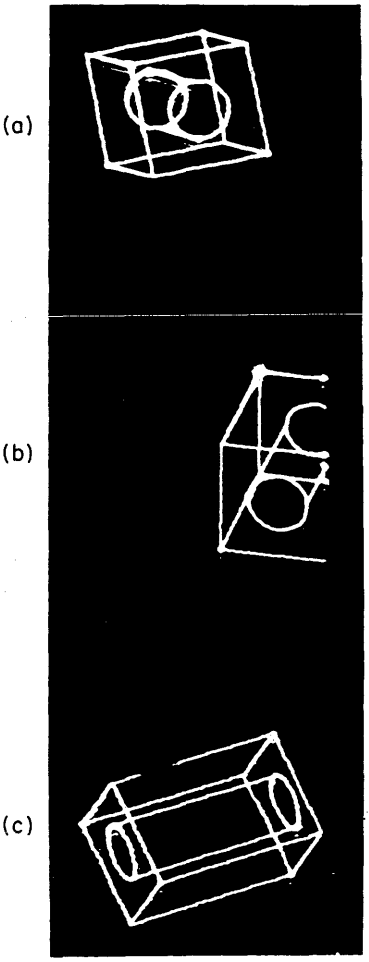


Figure 3. Figures Generated by Display System.
a) Orientation 1.
b) Orientation 2 showing "edging".
c) Orientation 3.

several configurations of the system in order to determine the best compromise between equipment cost, plotting speed and picture quality.

Figures 3 and 4 illustrate typical figures produced by the simulation routine. Figure 3 shows three orientations of a line drawing of a block with a hole through it. Part b) of this Figure demonstrates the technique used to prevent display when the edge of the scope is reached. Figure 4 depicts three different figures rotated about arbitrary axes.

The results of this simulation have shown that the system is capable of generating satisfactory, rotated, axonometric pictures using the Digital Differential Analyzer as the basic element of the line generator, and the cheaper Binary Rate Multipliers to perform the multiplications of the rotation matrix. The unit being built based on these results will have a basic clock rate between .5 and 1 megacycle, and when completed should provide an order of magnitude improvement in plotting speed over most present day point-plotting displays.

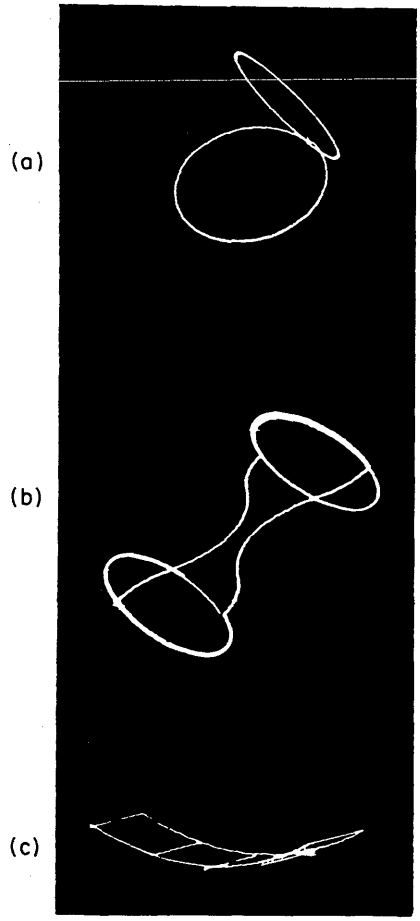


Figure 4. Figures Generated by Display System.
a) Generated by 7 bit DDA's.
b) Generated by 9 bit BRM's.
c) Generated by 9 bit DDA's.

DESIGN GOALS FOR M.I.T. DISPLAY SYSTEM

By providing straight-line and second-order curve generation, this display system is able to supply an order of magnitude increase in data rate over conventional scopes. Furthermore, since lines are specified by a minimum set of parameters a proportional savings in computing time and memory is achieved.

The novelty of this system, however, lies in its attempt to provide a new balance between

special purpose computing hardware and main frame computer load. It allows removal from the program of those functions which are simple, but must be done often. It also utilizes the display unit's abilities to perform certain operations easily (such as edge detection). The less frequent and more difficult tasks are left to the main frame, as for example, computation of the new values for the rotation matrix. But by keeping the display completely under the control of the computer, the program can bypass the special purpose equipment and assume any of the generation functions desired. With a human in the loop to act as monitor, the best compromise can be selected based on the needs of the operator.

Several extensions of this system are presently being investigated at the Electronic Systems Laboratory. The most promising of these are automatic light pen tracking and generation of an axonometric display of space curves, time consuming undertakings for a standard computer. Both of these should prove to be valuable in Computer-Aided Design. Although this display console is being built with the design studies in mind, it shows promise of becoming a versatile tool for research in the entire man-machine communication problem.

BIBLIOGRAPHY

1. *Investigations in Computer-Aided Design, Interim Report No. 1,* Report 8436-IR-1, Electronic Systems Laboratory, Massachusetts Institute of Technology, January 1961.
2. STOTZ, R. H., "Specialized Computer Equipment for Generation and Display of Three-Dimensional Curvilinear Figures," Report ESL-TM-167, Electronic Systems Laboratory, Massachusetts Institute of Technology, February 1963.
3. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Communication System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume).
4. RANDA, GLENN C., "Design of A Remote Display Console," Report ESL-R-132, Electronic Systems Laboratory, Massachusetts Institute of Technology, February 1962.
5. GURLEY, B. M., and WOODWARD, C. E., "Light-Pen Links Computer to Operator," *Electronics,* Vol. 32, No. 47, November 20, 1959.

SKETCHPAD
A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM*

Ivan E. Sutherland
*Consultant, Lincoln Laboratory***
Massachusetts Institute of Technology

I. INTRODUCTION

The Sketchpad system makes it possible for a man and a computer to converse rapidly through the medium of line drawings. Heretofore, most interaction between man and computers has been slowed down by the need to reduce all communication to written statements that can be typed; in the past, we have been writing letters to rather than conferring with our computers. For many types of communication, such as describing the shape of a mechanical part or the connections of an electrical circuit, typed statements can prove cumbersome. The Sketchpad system, by eliminating typed statements (except for legends) in favor of line drawings, opens up a new area of man-machine communication.

AN INTRODUCTORY EXAMPLE

To understand what is possible with the system at present let us consider using it to draw the hexagonal pattern in Figure 4. We will issue specific commands with a set of push buttons, turn functions on and off with switches, indicate position information and point to existing drawing parts with the light pen, rotate and magnify picture parts by turning knobs, and observe the drawing on the display system. This equipment as provided at Lincoln Laboratory's TX-2 computer[1] is shown in Figure 1. When our drawing is complete it may be inked on paper, as were all the drawings in this paper, by a PACE plotter.[15]

If we point the light pen at the display system and press a button called "draw," the computer will construct a straight line segment which stretches like a rubber band from the

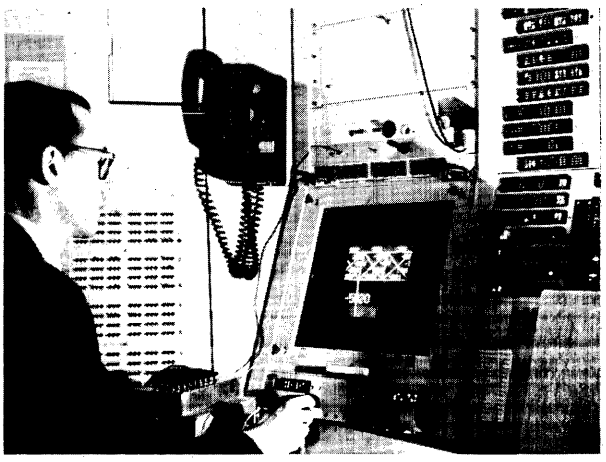


Figure 1. TX-2 operating area—Sketchpad in use. On the display can be seen part of a bridge similar to those of Figure 15. The Author is holding the light pen. The push buttons "draw," "move," etc., are on the box in front of the Author. Part of the bank of toggle switches can be seen behind the Author. The size and position of the part of tne total picture seen on the display are controlled by the four black knobs just above the tables.

* This paper is based in part on a thesis submitted tothe Department of Electrical Engineering, M.I.T., in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

** Operated with the support of the U.S. Army, Navy,and Air Force.

initial to the present location of the pen as shown in Figure 2. Additional presses of the button will produce additional lines, leaving the closed irregular hexagon shown in Figure 3A.

To make the hexagon regular, we can inscribe it in a circle. To draw the circle we place the light pen where the center is to be and press the button "circle center," leaving behind a center point. Now, choosing a point on the circle (which fixes the radius) we press the button "draw" again, this time getting a circle arc whose angular length only is controlled by light pen position as shown in Figure 2.

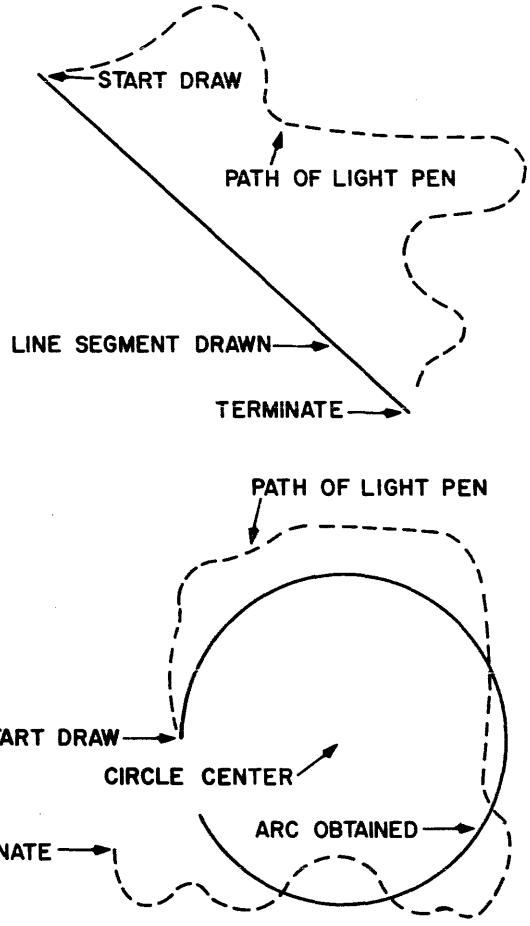


Figure 2. Steps for drawing straight lines and circle arcs.

Next we move the hexagon into the circle by pointing to a corner of the hexagon and pressing the button "move" so that the corner follows the light pen, stretching two rubber band line segments behind it. By pointing to the

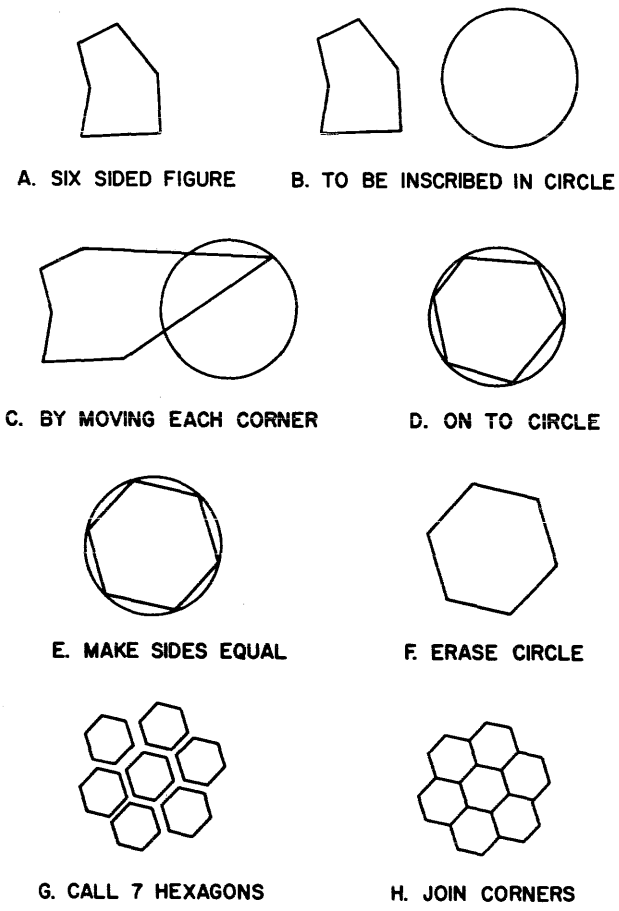


Figure 3. Illustrative example, see text.

circle and terminating, we indicate that the corner is to lie on the circle. Each corner is in this way moved onto the circle at roughly equal spacing as shown in Figure 3D.

We have indicated that the vertices of the hexagon are to lie on the circle, and they will remain on the circle throughout our further manipulations. If we also insist that the sides of the hexagon be of equal length, a regular hexagon will be constructed.

With Sketchpad we can say, in effect, make *this* line equal in length to *that* line, pointing to the lines with the light pen. The computer satisfies all existing conditions (if it is possible) whenever we turn on a toggle switch. This done, we have a complete regular hexagon inscribed in a circle. We can erase the entire circle by pointing to any part of it and pressing the "delete" button. The completed hexagon is shown in Figure 3F.

To make the hexagonal pattern in Figure 4 we wish to attach a large number of hexagons together by their corners, and so we designate the six corners of our hexagon as attachment points by pointing to each and pressing a button. We now file away the basic hexagon and begin work on a fresh "sheet of paper" by changing a switch setting. On the new sheet we assemble, by pressing a button to create each hexagon as an "instance" or subpicture, six hexagons around a central seventh in approximate position as shown in Figure 3G. A subpicture may be positioned with the light pen, rotated or scaled by turning the knobs, or fixed in position by a termination signal, but its internal shape is fixed.

By pointing to the corner of one hexagon, pressing a button, and then pointing to the corner of another hexagon, we can fasten those corners together, because these corners have been designated as attachment points. If we attach two corners of each outer hexagon to the appropriate corners of the inner hexagon, the seven are uniquely related, and the computer will reposition them as shown in Figure 3H. An entire group of hexagons, once assembled, can be treated as a symbol. An "instance" of the entire group can be called up on another "sheet of paper" as a subpicture and assembled with other groups or with single hexagons to make a very large pattern.

INTERPRETATION OF INTRODUCTORY EXAMPLE

In the introductory example above we used the light pen both to position parts of the drawing and to point to existing parts. We also saw in action the very general *subpicture, constraint,* and *definition copying* capabilities of the system.

Subpicture:

The original hexagon might just as well have been anything else: a picture of a transistor, a roller bearing, or an airplane wing. Any number of different symbols may be drawn, in terms of other simpler symbols if desired, and any symbol may be used as often as desired.

Constraint:

When we asked that the vertices of the hexagon lie on the circle we were making use of a basic relationship between picture parts that is built into the system. Basic relationships (atomic constraints) to make lines vertical, horizontal, parallel, or perpendicular; to make points lie on lines or circles; to make symbols appear upright, vertically above one another or be of equal size; and to relate symbols to other drawing parts such as points and lines have been included in the system. Specialized constraint types may be added as needed.

Definition Copying:

We made the sides of the hexagon be equal in length by pressing a button while pointing to the side in question. Had we defined a composite operation such as to make two lines both parallel and equal in length, we could have applied it just as easily.

IMPLICATIONS OF INTRODUCTORY EXAMPLE

As we have seen, a Sketchpad drawing is entirely different from the trail of carbon left on a piece of paper. Information about how the drawing is tied together is stored in the computer as well as the information which gives the drawing its particular appearance. Since the drawing is tied together, it will keep a useful appearance even when parts of it are moved. For example, when we moved the corners of the hexagon onto the circle, the lines next to each corner were automatically moved so that the closed topology of the hexagon was preserved. Again, since we indicated that the corners of the hexagon were to lie on the circle, they remained on the circle throughout our further manipulations.

As well as storing how the various parts of the drawing are related, Sketchpad stores the structure of the subpictures used. For example, the storage for the hexagonal pattern of Figure 4 indicates that this pattern is made of smaller patterns which are in turn made of smaller patterns which are composed of single hexagons. If the master hexagon is changed, the entire appearance but not the structure of the hexagonal pattern will be changed. For example, if we change the basic hexagon into a semicircle, the fish scale pattern shown in Figure 4 instantly results.

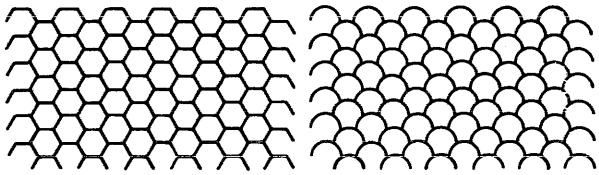


Figure 4. Hexagonal lattice with half hexagon and semicircle as basic elements.

SKETCHPAD AND THE DESIGN PROCESS

Construction of a drawing with Sketchpad is *itself* a model of the design process. The locations of the points and lines of the drawing model the variables of a design, and the geometric constraints applied to the points and lines of the drawing model the design constraints which limit the values of design variables. The ability of Sketchpad to satisfy the geometric constraints applied to the parts of a drawing models the ability of a good designer to satisfy all the design conditions imposed by the limitations of his materials, cost, etc. In fact, since designers in many fields produce nothing themselves but a drawing of a part, design conditions may well be thought of as applying to the drawing of a part rather than to the part itself. When such design conditions are added to Sketchpad's vocabulary of constraints, the computer will be able to assist a user not only in arriving at a nice looking drawing, but also in arriving at a sound design.

PRESENT USEFULNESS

As more and more applications have been made, it has become clear that the properties of Sketchpad drawings make them most useful in four broad areas:

For Storing and Updating Drawings:

Each time a drawing is made, a description of that drawing is stored in the computer in a form that is readily transferred to magnetic tape. A library of drawings will thus develop, parts of which may be used in other drawings at only a fraction of the investment of time that was put into the original drawing.

For Gaining Scientific or Engineering Understanding of Operations That Can Be Described Graphically:

A drawing in the Sketchpad system may contain explicit statements about the relations between its parts so that as one part is changed the implications of this change become evident throughout the drawing. For instance, Sketchpad makes it easy to study mechanical linkages, observing the path of some parts when others are moved.

As a Topological Input Device for Circuit Simulators, etc.:

Since the storage structure of Sketchpad reflects the topology of any circuit or diagram, it can serve as an input for many network or circuit simulating programs. The additional effort required to draw a circuit completely from scratch with the Sketchpad system may well be recompensed if the properties of the circuit are obtainable through simulation of the circuit drawn.

For Highly Repetitive Drawings:

The ability of the computer to reproduce any drawn symbol anywhere at the press of a button, and to recursively include subpictures within subpictures makes it easy to produce drawings which are composed of huge numbers of parts all similar in shape.

II. RING STRUCTURE

The basic n-component element structure described by Ross[10] has been somewhat expanded in the implementation of Sketchpad so that all references made to a particular n-component element or block are collected together by a string of pointers which originates within that block. For example, not only may the end points of a line segment be found by following pointers in the line block (n-component element), but also all the line segments which terminate on a particular point may be found by following a string of pointers which starts within the point block. This string of pointers closes on itself; the last pointer points back to the first, hence the name "ring." The ring points both ways to make it easy to find both the next and the previous member of the ring in case, as when deleting, some change must be made to them.

BASIC OPERATIONS

The basic ring structure operations are:

1. Inserting a new member into a ring at

some specified location on it, usually first or last.

2. Removing a member from a ring.
3. Putting all the members of one ring, in order, into another at some specified location in it, usually first or last.
4. Performing some auxiliary operation on each member of a ring in either forward or reverse order.

These basic ring structure operations are implemented by short sections of program defined as MACRO instructions in the compiler language. By suitable treatment of zero and one member rings, the basic programs operate without making special cases.

Subroutines are used for setting up new n-component elements in free spaces in the storage structure. As parts of the drawing are deleted, the registers which were used to represent them become free. New components are set up at the end of the storage area, lengthening it, while free blocks are allowed to accumulate. Garbage collection periodically compacts the storage structure by removal of the free blocks.

GENERIC STRUCTURE, HIERARCHIES

The main part of Sketchpad can perform basic operations on any drawing part, calling for help from routines specific to particular types of parts when that is necessary. For example, the main program can show any part on the display system by calling the appropriate display subroutine. The big power of the clear-cut separation of the general and the specific is that it is easy to change the details of specific parts of the program to get quite different results without any need to change the general parts.

In the data storage structure the separation of general and specific is accomplished by collecting all things of one type together in a ring under a generic heading. The generic heading contains all the information which makes this type of thing different from all other types of things. Thus the data storage structure itself contains all the specific information. The generic blocks are further gathered together under super-generic or generic-generic blocks, as shown in Figure 5.

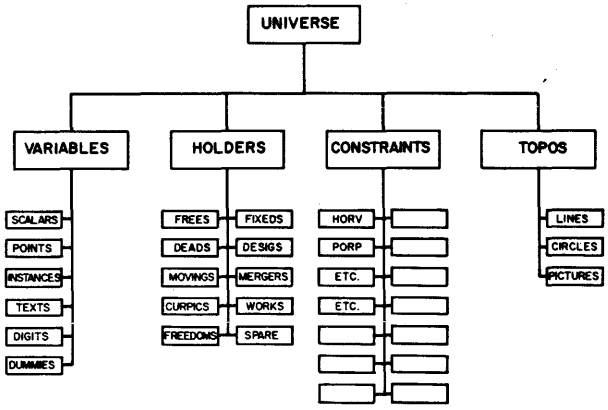


Figure 5. Generic structure. The n-component elements for each point or line, etc., are collected under the generic blocks "lines," "points," etc., shown.

EXPANDING SKETCHPAD

Addition of new types of things to the Sketchpad system's vocabulary of picture parts requires only the construction of a new generic block (about 20 registers) and the writing of appropriate subroutines for the new type. The subroutines might be easy to write, as they usually are for new constraints, or difficult to write, as for adding ellipse capability, but at least a finite, well-defined task faces one to add a new ability to the system. Without a generic structure it would be almost impossible to add the instructions required to handle a new type of element.

III. LIGHT PEN

In Sketchpad the light pen* is time shared between the functions of coordinate input for positioning picture parts on the drawing and demonstrative input for pointing to existing picture parts to make changes. Although almost any kind of coordinate input device could be used instead of the light pen for positioning, the demonstrative input uses the light pen optics as a sort of analog computer to remove from consideration all but a very few picture parts which happen to fall within its field of view, saving considerable program time. Drawing systems using storage display devices of the Memotron type may not be practical because of the loss of this analog computation feature.

* The reader unacquainted with light pens should refer to the paper on Man-Machine Console Facilities by Stotz[12] in this issue.

PEN TRACKING

To initially establish pen tracking,* the Sketchpad user must inform the computer of an initial pen location. This has come to be known as "inking-up" and is done by "touching" any existing line or spot on the display, whereupon the tracking cross appears. If no picture has yet been drawn, the letters INK are always displayed for this purpose. Sketchpad uses loss of tracking as a "termination signal" to stop drawing. The user signals that he is finished drawing by flicking the pen too fast for the tracking program to follow.

DEMONSTRATIVE USE OF PEN

During the 90% of the time that the light pen and display system are free from the tracking chore, spots are very rapidly displayed to exhibit the drawing being built, and thus the lines and circles of the drawing appear. The light pen is sensitive to these spots and reports any which fall within its field of view. Thus, a table of the picture parts seen by the light pen is assembled during each complete display cycle. At the end of a display cycle this table contains all the picture parts that could even remotely be considered as being "aimed at."

The one-half inch diameter field of view of the light pen, although well suited to tracking, is relatively large for pointing. Therefore, the Sketchpad system will reject any seen part which is further from the center of the light pen than some small minimum distance; about 1/8 inch was found to be suitable. For every kind of picture part some method must be provided for computing its distance from the light pen center or indicating that this computation cannot be made.

After eliminating all parts seen by the pen which lie outside the smaller effective field of view, the Sketchpad system considers objects topologically related to the ones actually seen. End points of lines and attachment points of instances (subpictures) are especially important. One can thus aim at the end point of a line even though only the line is displayed. Figure 6 outlines the various regions within which the pen must lie to be considered aimed at a line segment, a circle arc, their end points, or their intersection.

PSEUDO PEN LOCATION

When the light pen is aimed at a picture part, the exact location of the light pen is ignored in favor of a "pseudo pen location" exactly on the part aimed at. If no object is aimed at, the pseudo pen location is taken to be the actual pen location. The pseudo pen location is displayed as a bright dot which is used as the "point of the pencil" in all drawing operations. As the light pen is moved into the areas outlined in Figure 6 the dot will lock onto the existing parts of the drawing, and any moving picture parts will jump to their new locations as the pseudo pen location moves to lie on the appropriate picture part.

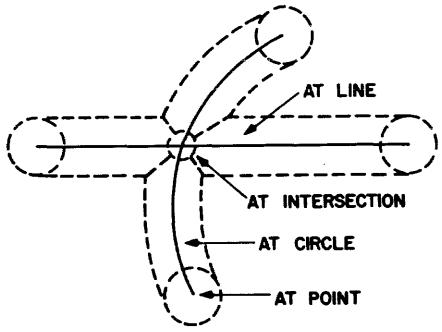


Figure 6. Areas in which pen must lie to "aim at" existing drawing parts (solid lines).

With just the basic drawing creation and manipulation functions of "draw," "move," and "delete," and the power of the pseudo pen location and demonstrative language programs, it is possible to make fairly extensive drawings. Most of the constructions normally provided by straight edge and compass are available in highly accurate form. Most important, however, the pseudo pen location and demonstrative language give the means for entering the topological properties of a drawing into the machine.

IV. DISPLAY GENERATION

The display system, or "scope," on the TX-2 is a ten bit per axis electrostatic deflection system able to display spots at a maximum rate of about 100,000 per second. The coordinates of the spots which are to be seen on the display are stored in a large table so that computation and display may proceed independently. If, instead of displaying each spot successively, the

display program displays them in a random order or with interlace, the flicker of the display is reduced greatly.

MARKING OF DISPLAY FILE

Of the 36 bits available to store each display spot in the display file, 20 give the coordinates of that spot for the display system, and the remaining 16 give the address of the n-component element which is responsible for adding that spot to the display. Thus, all the spots in a line are tagged with the ring structure address of that line, and all the spots in an instance (subpicture) are tagged as belonging to that instance. The tags are used to identify the particular part of the drawing being aimed at by the light pen.

If a part of the drawing is being moved by the light pen, its display spots will be recomputed as quickly as possible to show it in successive positions. The display spots for such moving parts are stored at the end of the display file so that the display of the many nonmoving parts need not be disturbed. Moving parts are made invisible to the light pen.

MAGNIFICATION OF PICTURES

The shaft position encoder knobs below the scope (see Figure 1) are used to tell the program to change the display scale factor or the portion of the page displayed. The range of magnification of 2000 available makes it possible to work, in effect, on a 7-inch square portion of a drawing about ¼ mile on a side.

For a magnified picture, Sketchpad computes which portion(s) of a curve will appear on the display and generates display spots for those portions only. The "edge detection" problem is the problem of finding suitable end points for the portion of a curve which appears on the display.

In concept the edge detection problem is trivial. In terms of program time for lines and circles the problem is a small fraction of the total computational load of the system, but in terms of program logical complexity the edge detection problem is a difficult one. For example, the computation of the intersection of a circle with any of the edges of the scope is easy, but computation of the intersection of a circle with all four edges may result in as many as eight intersections, some pairs of which may be identical, the scope corners. Now which of these intersections are actually to be used as starts of circle arcs?

LINE AND CIRCLE GENERATION

All of Sketchpad's displays are generated from straight line segments, circle arcs, and single points. The generation of the lines and circles is accomplished by means of the difference equations:

$$x_i = x_{i-1} + \Delta x \qquad y_i = y_{i-1} + \Delta y \tag{1}$$

for lines, and

$$\begin{aligned} x_i &= x_{i-2} + \frac{2}{R}(y_{i-1} - y_c) \\ y_i &= y_{i-2} - \frac{2}{R}(x_{i-1} - x_c) \end{aligned} \tag{2}$$

for circles, where subscripts i indicate successive display spots, subscript c indicates the circle center, and R is the radius of the circle in Scope Units. In implementing these difference equations in the program, the fullest possible use is made of the coordinate arithmetic capability of the TX-2 so that both the x and y equation computations are performed in parallel on 18 bit subwords. Even so, about ¾ of the total Sketchpad computation time is spent in line and circle generation. A vector and circle generating display would materially reduce the computational load of Sketchpad.

For computers which do only one addition at a time, the difference equations:

$$\begin{aligned} x_i &= x_{i-1} + \frac{1}{R}(y_{i-1} - y_c) \\ y_i &= y_{i-1} - \frac{1}{R}(x_i - x_c) \end{aligned} \tag{3}$$

should be used to generate circles. Equations (3) approximate a circle well enough and are known to close exactly both in theory and when implemented, because the x and y equations are dissimilar.

DIGITS AND TEXT

Text, to put legends on a drawing, is displayed by means of special tables which indicate the locations of line and circle segments to make up the letters and numbers. Each piece of text appears as a single line of not more

than 36 equally spaced characters which can be changed by typing. Digits to display the value of an indicated scalar at any position and in any size and rotation are formed from the same type face as text. It is possible to display up to five decimal digits with sign; binary to decimal conversion is provided, and leading zeros are suppressed.

Subpictures, whose use was seen in the introductory example above, are each represented in storage as a single n-component element. A subpicture is said to be an "instance" of its "master picture." To display an instance, all of the lines, text, etc. of its master picture must be shown in miniature on the display. The instance display program makes use of the line, circle, number, and text display programs and *itself* to expand the internal structure of the instance.

DISPLAY OF ABSTRACTIONS

The usual picture for human consumption displays only lines, circles, text, digits, and instances. However, certain very useful abstractions which give the drawing the properties desired by the user are represented in the ring structure storage. For example, the fact that the start and end points of a circle arc should be equidistant from the circle's center point is represented in storage by a "constraint" block. To make it possible for a user to manipulate these abstractions, each abstraction must be able to be seen on the display if desired. Not only does displaying abstractions make it possible for the human user to know that they exist, but also makes it possible for him to aim at them with the light pen and, for example, erase them. To avoid confusion, the display for particular types of objects may be turned on or off selectively by toggle switches. Thus, for example, one can turn on display of constraints as well as or instead of the lines and circles which are normally seen.

If their selection toggle switch is on, constraints are displayed as shown in Figure 7. The central circle and code letter are located at the average location of the variables constrained. The four arms of a constraint extend from the top, right side, bottom, and left side of the circle to the first, second, third, and fourth variables constrained, respectively. If fewer than four variables are constrained, excess arms are omitted. In Figure 7 the constraints are shown applied to "dummy variables," each of which shows as an X.

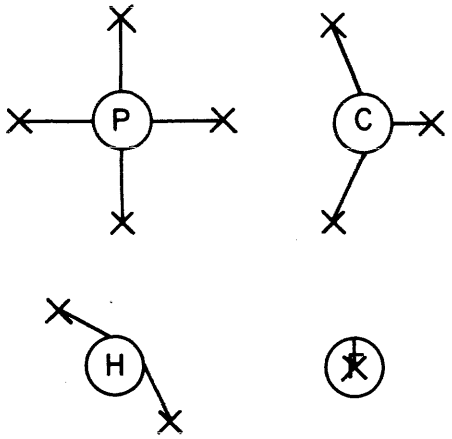


Figure 7. Display of constraints.

Another abstraction that can be displayed if desired is the value of a set of digits. For example, in Figure 8 are shown three sets of digits all displaying the same scalar value, -5978. The digits themselves may be moved, rotated, or changed in size, without changing the value displayed. If we wish to change the value, we point at its abstract display, the # seen in Figure 8. The three sets of digits in Figure 8 all display the same value, as indicated by the lines connecting them to the #; changing this value would make all three sets of digits change. Constraints may be applied independently to either the position of the digits or their value as indicated by the two constraints in the figure.

V. RECURSIVE FUNCTIONS

In the process of making the Sketchpad system operate, a few very general functions were developed which make no reference at all to the specific types of entities on which they oper-

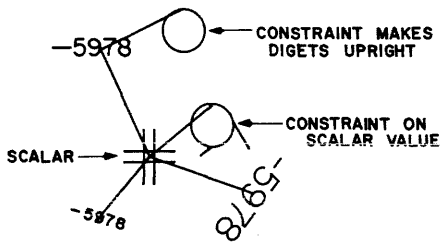


Figure 8. Three sets of digits displaying the same scalar value.

ate. These general functions give the Sketchpad system the ability to operate on a wide range of problems. The motivation for making the functions as general as possible came from the desire to get as much result as possible from the programming effort involved. For example, the general function for expanding instances makes it possible for Sketchpad to handle any fixed geometry subpicture. The power obtained from the small set of generalized functions in Sketchpad is one of the most important results of the research.

In order of historical development, the recursive functions in use in the Sketchpad system are:

1. Expansion of instances, making it possible to have subpictures within subpictures to as many levels as desired.
2. Recursive deletion, whereby removal of certain picture parts will remove other picture parts in order to maintain consistency in the ring structure.
3. Recursive merging, whereby combination of two similar picture parts forces combination of similarly related other picture parts, making possible application of complex definitions to an object picture.

RECURSIVE DELETING

If a thing upon which other things depend is deleted, the dependent things must be deleted also. For example, if a point is to be deleted, all lines which terminate on the point must also be deleted. Otherwise, since the n-component elements for lines contain no positional information, where would these lines end? Similarly, deletion of a variable requires deletion of all constraints on that variable; a constraint must have variables to act on.

RECURSIVE MERGING

If two things of the same type which are independent are merged, a single thing of that type results, and all things which depended on either of the merged things depend on the result of the merger.* For example, if two points are merged, all lines which previously terminated on either point now terminate on the single resulting point. In Sketchpad, if a thing is being moved with the light pen and the termination flick of the pen is given while aiming at another thing of the same type, the two things will merge. Thus, if one moves a point to another point and terminates, the points will merge, connecting all lines which formerly terminated on either. This makes it possible to draw closed polygons.

If two things of the same type which do depend on other things are merged, the things depended on by one will be forced to merge, respectively, with the things depended on by the other. The result of merging two dependent things depends, respectively, on the results* of the mergers it forces.* For example, if two lines are merged, the resultant line must refer to only two end points, the results of merging the pairs of end points of the original lines. All lines which terminated on any of the four original end points now terminate on the appropriate one of the remaining pair. More important and useful, all constraints which applied to any of the four original end points now apply to the appropriate one of the remaining pair. This makes it possible to speak of line segments as being parallel even though (because line segments contain no numerical information to be constrained) the parallelism constraint must apply to their end points and not to the line segments themselves. If we wish to make two lines both parallel and equal in length, the steps outlined in Figure 9 make it possible. More obscure relationships between dependent things may be easily defined and applied. For example, constraint complexes can be defined to make line segments be collinear, to make a line be tangent to a circle, or to make the values represented by two sets of digits be equal.

RECURSIVE DISPLAY OF INSTANCES

The block of registers which represents an instance is remarkably small considering that it may generate a display of any complexity. For the purposes of display, the instance block makes reference to its master picture. The instance will appear on the display as a figure geometrically similar to its master picture at a location, size, and rotation indicated by the four numbers which constitute the "value" of the instance. The value of an instance is considered numerically as a four dimensional vector. The

* The "result" of a merger is a single thing of the same type as the merged things.

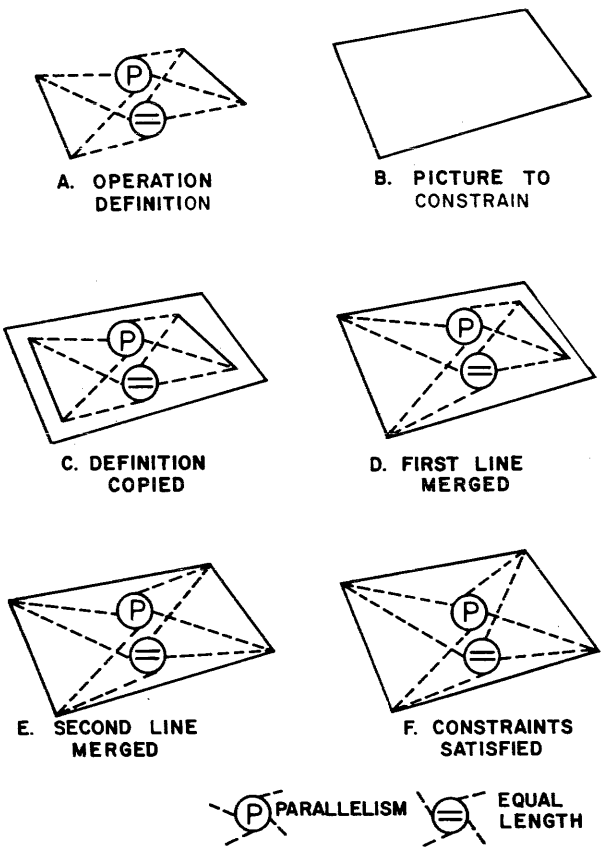


Figure 9. Applying a two-constraint definition to turn a quadrilateral into a parallelogram.

components of this vector are the coordinates of the center of the instance and its actual size as it appears on the drawing times the sine and cosine of the rotation angle involved.

In displaying an instance of a picture, reference is made to the master picture to find out what picture parts are to be shown. The master picture referred to may contain instances, however, requiring further reference, and so on until a picture is found which contains no instances. At each stage in the recursion, any picture parts displayed must be relocated so that they will appear at the correct position, size and rotation on the display. Thus, at each stage of the recursion, some transformation is applied to all picture parts before displaying them. If an instance is encountered, the transformation represented by its value must be adjoined to the existing transformation for display of parts within it. When the expansion of an instance within an instance is finished, the transformation must be restored for continuation at the higher level.

ATTACHERS AND INSTANCES

Many symbols must be integrated into the rest of the drawing by attaching lines to the symbols at appropriate points, or by attaching the symbols directly to each other. For example, circuit symbols must be wired up, geometric patterns made by fitting shapes together, or mechanisms composed of links tied together appropriately. An instance may have any number of attachment points, and a point may serve as attacher for any number of instances. The light pen has the same affinity for the attachers of an instance that it has for the end point of a line.

An "instance-point" constraint, shown with code T in Figure 10C, is used to relate an instance to each of its attachment points. An instance-point constraint is satisfied only when the point bears the same relationship to the instance that a master point in the master picture for that instance bears to the master picture coordinate system.

Any point may be an attacher of an instance, but the point must be designated as an attacher in the master drawing of the instance. For example, when one first draws a resistor, the ends of the resistor must be designated as attachers if wiring is to be attached to instances of it. At each level of building complex pictures, the attachers must be designated anew. Thus of the three attachers of a transistor it is possible to select one or two to be the attachers of a flip-flop.

VI. BUILDING A DRAWING, THE COPY FUNCTION

At the start of the Sketchpad effort certain ad hoc drawing functions were programmed as the atomic operations of the system. Each such operation, controlled by a push button, creates in the ring structure a specific set of new drawing parts. For example, the "draw" button creates a line segment and two new end points (unless the light pen happens to be aimed at a point in which case only one new point need be created). Similarly, there are atomic operations for drawing circles, applying a horizontal or vertical constraint to the end points of a line aimed at, and for adding a "point-on-line" constraint whenever a point is moved onto a line and left there.

The atomic operations described above make it possible to create in the ring structure new picture components and relate them topologically. The atomic operations are, of course, limited to creating points, lines, circles, and two or three types of constraints. Since implementation of the copy function it has become possible to create in the ring structure any predefined combination of picture parts and constraints at the press of a button. The recursive merging function makes it possible to relate the copied set of picture parts to any existing parts. For example, if a line segment and its two end points are copied into the object picture, the action of the "draw" button may be exactly duplicated in every respect. Along with the copied line, however, one might copy as well a constraint, Code H, to make the line horizontal as shown in Figure 10A, or two constraints to make the line both horizontal and three inches long, or any other variation one cares to put into the ring structure to be copied.

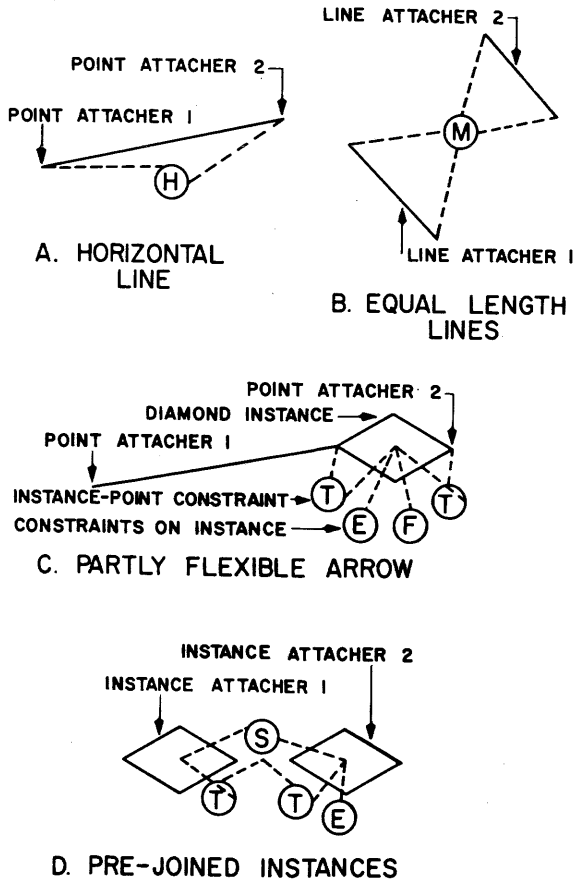


Figure 10. Definition pictures to be copied, see text.

When one draws a definition picture to be copied, certain portions of it to be used in relating it to other object picture parts are designated as "attachers." Anything at all may be designated: for example, points, lines, circles, text, even constraints! The rules used for combining points when the "draw" button is pressed are generalized so that:

> For copying a picture, the last-designated attacher is left moving with the light pen. The next-to-last-designated attacher is recursively merged with whatever object the pen is aimed at when the copying occurs, if that object is of like type. Previously designated attachers are recursively merged with previously designated object picture parts, if of like type, until either the supply of designated attachers or the supply of designated object picture parts is exhausted. The last-designated attacher may be recursively merged with any other object of like type when the termination flick is given.

Normally only two designated attachers are used because it is hard to keep track of additional ones.

If the definition picture consists of two line segments, their four end points, and a constraint, Code M, on the points which makes the lines equal in length, with the two lines designated as attachers as shown in Figure 10B, copying enables the user to make any two lines equal in length. If the pen is aimed at a line when "copy" is pushed, the first of the two copied lines merges with it (taking its position and never actually being seen). The other copied line is left moving with the light pen and will merge with whatever other line the pen is aimed at when termination occurs. Since merging is recursive, the copied equal-length constraint, Code M, will apply to the end points of the desired pair of object picture lines.

COPYING INSTANCES

As we have seen above, the internal structure of an instance is entirely fixed. The internal structure of a copy, however, is entirely variable. An instance always retains its identity as a single part of the drawing; one can only delete an entire instance. Once a definition picture is copied, however, the copy loses all identity as a unit; individual parts of it may be deleted at will.

One might expect that there was intermediate ground between the fixed-internal-structure instance and the loose-internal-structure copy. One might wish to produce a collection of picture parts, some of which were fixed internally and some of which were not. *The entire range of variation between the instance and the copy can be constructed by copying instances.*

For example, the arrow shown in Figure 10C can be copied into an object picture to result in a fixed-internal-structure diamond arrowhead with a flexible tail. As the definition in Figure 10C is set up, drawing diamond-arrowheaded lines is just like drawing ordinary lines. One aims the light pen where the tail is to end, presses "copy," and moves off with an arrowhead following the pen. The diamond arrowhead in this case will not rotate (constraint Code E), and will not change size (constraint Code F).

Copying pre-joined instances can produce vast numbers of joined instances very easily. For example, the definition in Figure 10D, when repetitively copied, will result in a row of joined, equal size (constraint Code S) diamonds. In this case the instances themselves are attachers. Although each press of the "copy" button copies two new instances into the object picture, one of these is merged with the last instance in the growing row. In the final row, therefore, each instance carries all constraints which are applied to either of the instances in the definition. This is why only one of the instances in Figure 10D carries the erect constraint, Code E.

VII. CONSTRAINT SATISFACTION

The major feature which distinguishes a Sketchpad drawing from a paper and pencil drawing is the user's ability to specify to Sketchpad mathematical conditions on already drawn parts of his drawing which will be automatically satisfied by the computer to make the drawing take the exact shape desired. The process of fixing up a drawing to meet new conditions applied to it after it is already partially complete is very much like the process a designer goes through in turning a basic idea into a finished design. As new requirements on the various parts of the design are thought of, small changes are made to the size or other properties of parts to meet the new conditions. By making Sketchpad able to find new values for variables which satisfy the conditions imposed, it is hoped that designers can be relieved of the need of much mathematical detail. The effort expended in making the definition of constraint types as general as possible was aimed at making design constraints as well as geometric constraints equally easy to add to the system.

DEFINITION OF A CONSTRAINT TYPE

Each constraint type is entered into the system as a generic block indicating the various properties of that particular constraint type. The generic block tells how many variables are constrained, which of these variables may be changed in order to satisfy the constraint, how many degrees of freedom are removed from the constrained variables, and a code letter for human reference to this constraint type.

The definition of what a constraint type does is a subroutine which will compute, for the existing values of the variables of a particular constraint of that type, the error introduced into the system by that particular constraint. For example, the defining subroutine for making points have the same x coordinate (to make a line between them vertical) computes the difference in their x coordinates. What could be simpler? The computed error is a scalar which the constraint satisfaction routine will attempt to reduce to zero by manipulation of the constrained variables. The computation of the error may be non-linear or time dependent, or it may involve parameters not a part of the drawing such as the setting of toggle switches, etc.

When the one pass method of satisfying constraints to be described later on fails, the Sketchpad system falls back on the reliable but slow method of relaxation[11] to reduce the errors indicated by various computation subroutines to smaller and smaller values. For simple constructions such as the hexagon illustrated in Figure 3, the relaxation procedure is sufficiently fast to be useful. However, for complex systems of variables, especially directly connected instances, relaxation is unacceptably slow. Fortunately it is for just such directly connected instances that the one pass method shows the most striking success.

ONE PASS METHOD

Sketchpad can often find an order in which the variables of a drawing may be re-evaluated to completely satisfy all the conditions on them in just one pass. For the cases in which the one pass method works, it is far better than relaxation: it gives correct answers at once; relaxation may not give a correct solution in any finite time. Sketchpad can find an order in which to re-evaluate the variables of a drawing for most of the common geometric constructions. Ordering is also found easily for the mechanical linkages shown in Figures 13 and 14. Ordering cannot be found for the bridge truss problem in Figure 15.

The way in which the one pass method works is simple in principle and was easy to implement as soon as the nuances of the ring structure manipulations were understood. To visualize the one pass method, consider the variables of the drawing as places and the constraints relating variables as passages through which one might pass from one variable to another. Variables are adjacent to each other in the maze formed by the constraints if there is a single constraint which constrains them both. Variables are totally unrelated if there is no path through the constraints by which to pass from one to the other.

Suppose that some variable can be found which has so few constraints applying to it that it can be re-evaluated to completely satisfy all of them. Such a variable we shall call a "free" variable. As soon as a variable is recognized as free, the constraints which apply to it are removed from further consideration, because the free variable can be used to satisfy them. Removing these constraints, however, may make adjacent variables free. Recognition of these new variables as free removes further constraints from consideration and may make other adjacent variables free, and so on throughout the maze of constraints. The manner in which freedom spreads is much like the method used in Moore's algorithm[8] to find the shortest path through a maze. Having found that a collection of variables is free, Sketchpad will re-evaluate them in reverse order, saving the first-found free variable until last. In re-evaluating any particular variable, Sketchpad uses only those constraints which were present when that variable was found to be free.

VIII. EXAMPLES AND CONCLUSIONS

The examples in this section were all taken from the library tape and thus serve to illustrate not only how the Sketchpad system can be used, but also how it actually has been used so far. We conclude from these examples that Sketchpad drawings can bring invaluable understanding to a user. For drawings where motion of the drawing, or analysis of a drawn problem is of value to the user, Sketchpad excels. For highly repetitive drawings or drawings where accuracy is required, Sketchpad is sufficiently faster than conventional techniques to be worthwhile. For drawings which merely communicate with shops, it is probably better to use conventional paper and pencil.

PATTERNS

The instance facility enables one to draw any symbol and duplicate its appearance anywhere on an object drawing at the push of a button. This facility made the hexagonal pattern we saw in Figure 4 easy to draw. It took about one half hour to generate 900 hexagons, including the time taken to figure out how to do it. Plotting them takes about 25 minutes. The drafting department estimated it would take two days to produce a similar pattern.

The instance facility also made it easy to produce long lengths of the zig-zag pattern shown in Figure 11. As the figure shows, a single "zig" was duplicated in multiples of five and three, etc. Five hundred zigs were generated in a single row. Four such rows were plotted one-half inch apart to be used for producing a printed circuit delay line. Total time taken was about 45 minutes for constructing the figure and about 15 minutes to plot it.

A somewhat less repetitive pattern to be used for encoding the time in a digital clock is shown in Figure 12. Each cross in the figure marks the position of a hole. The holes are placed so that a binary coded decimal (BCD) number will in-

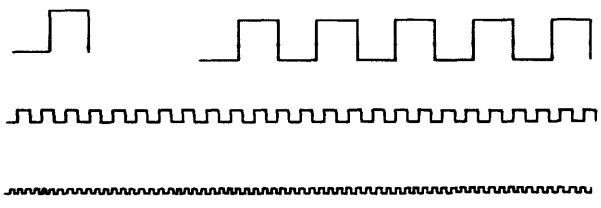


Figure 11. Zig-Zag for delay line.

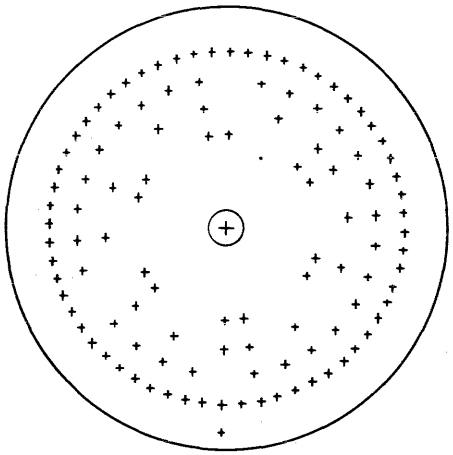


Figure 12. Binary coded decimal encoder for clock. Encoder was plotted exactly 12 inches in diameter for direct use as a layout.

dicate the time. Total time for placing crosses was 20 minutes, most of which was spent trying to interpret a pencil sketch of their positions.

LINKAGES

By far the most interesting application of Sketchpad so far has been drawing and moving linkages. The ability to draw and then move linkages opens up a new field of graphical manipulation that has never before been available. It is remarkable how even a simple linkage can generate complex motions. For example, the linkage of Figure 13 has only three moving parts. In this linkage a central ⊥ link is suspended between two links of different lengths. As the shorter link rotates, the longer one oscillates as can be seen in the multiple exposure. The ⊥ link is not shown in Figure 13 so that the motion of four points on the upright part of the ⊥ may be seen. These are the four curves at the top of the figure.

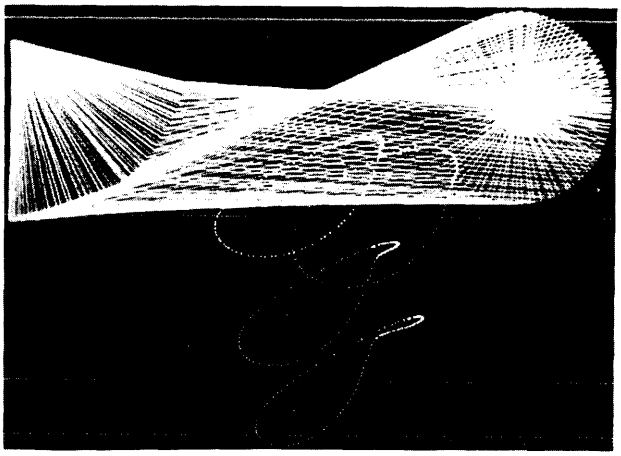


Figure 13. Three bar linkage. The paths of four points on the central link are traced. This is a 15 second time exposure of a moving Sketchpad drawing.

To make the three bar linkage, an instance shaped like the ⊥ was drawn and given 6 attachers, two at its joints with the other links and four at the places whose paths were to be observed. Connecting the ⊥ shaped subpicture onto a linkage composed of three lines with fixed length created the picture shown. The driving link was rotated by turning a knob below the scope. Total time to construct the linkage was less than 5 minutes, but over an hour was spent playing with it.

A linkage that would be difficult to build physically is shown in Figure 14 A. This link-

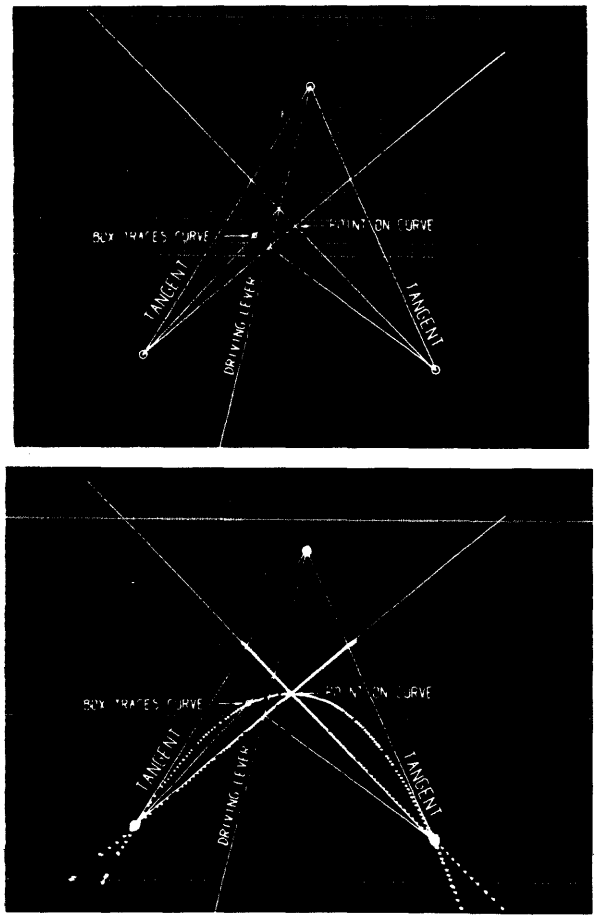


Figure 14. Conic drawing linkage. As the "driving lever" is moved, the point shown with a box around it (in A) traces a conic section. This conic can be seen in the time exposure (B).

age is based on the complete quadrilateral. The three circled points and the two lines which extend out of the top of the picture to the right and left are fixed. Two moving lines are drawn from the lower circled points to the intersections of the long fixed lines with the driving lever. The intersection of these two moving lines (one must be extended) has a box around it. It can be shown theoretically that this linkage produces a conic section which passes through the place labeled "point on curve" and is tangent to the two lines marked "tangent." Figure 14 B shows a time exposure of the moving point in many positions. At first, this linkage was drawn and working in 15 minutes. Since then we have rebuilt it time and again until now we can produce it from scratch in about 3 minutes.

DIMENSION LINES

To make it possible to have an absolute scale in drawings, a constraint is provided which forces the value displayed by a set of digits to indicate the distance between two points on the drawing. This distance-indicating constraint is used to make the number in a dimension line correspond to its length. Putting in a dimension line is as easy as drawing any other line. One points to where one end is to be left, copies the definition of the dimension line by pressing the "copy" button, and then moves the light pen to where the other end of the dimension line is to be. The first dimension line took about 15 minutes to construct, but that need never be repeated since it is a part of the library.

BRIDGES

One of the largest untapped fields for application of Sketchpad is as an input program for other computation programs. The ability to place lines and circles graphically, when coupled with the ability to get accurately computed results pictorially displayed, should bring about a revolution in computer application. By using Sketchpad's relaxation procedure we were to demonstrate analysis of the force distribution in the members of a pin connected truss.

A bridge is first drawn with enough constraints to make it geometrically accurate. These constraints are then deleted and each member is made to behave like a bridge beam. A bridge beam is constrained to maintain constant length, but any change in length is indicated by an associated number. Under the assumption that each bridge beam has a cross-sectional area proportional to its length, the numbers represent the forces in the beams. The basic bridge beam definition (consisting of two constraints and a number) may be copied and applied to any desired line in a bridge picture by pointing to the line and pressing the "copy" button.

Having drawn a basic bridge shape, one can experiment with various loading conditions and supports to see what the effect of making minor modifications is. For example, an arch bridge is shown in Figure 15 supported both as a three-hinged arch (two supports) and as a cantilever (four supports). For nearly identical loading conditions the distribution of forces is markedly different in these two cases.

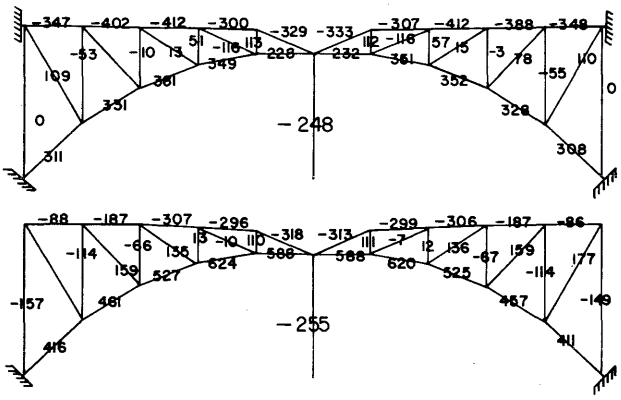


Figure 15. Cantilever and arch bridges. The numbers indicate the forces in the various members as computed by Sketchpad. Central load is not exactly vertical

ARTISTIC DRAWINGS

Sketchpad need not be applied exclusively to engineering drawings. For example, the girl "Nefertite" shown in Figure 16 can be made to wink by changing which of the three types of eyes is placed in position on her otherwise eyeless face. In the same way that linkages can be made to move, a stick figure could be made to pedal a bicycle or Nefertite's hair could be made to swing. The ability to make moving drawings suggests that Sketchpad might be used for making animated cartoons.

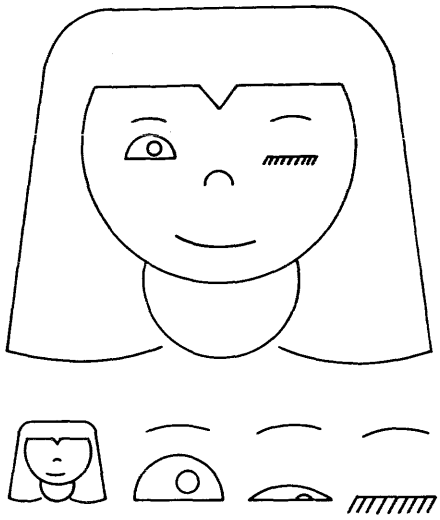


Figure 16. Winking girl, "Nefertite," and her component parts.

ELECTRICAL CIRCUIT DIAGRAMS

Unfortunately, electrical circuits require a great many symbols which have not yet been drawn properly with Sketchpad and therefore are not in the library. After some time is spent working on the basic electrical symbols it may be easier to draw circuits. So far, however, circuit drawing has proven difficult.

The circuits of Figure 17 are parts of an analog switching scheme. You can see in the figure that the more complicated circuits are made up of simpler symbols and circuits. It is very difficult, however, to plan far enough ahead to know what composites of circuit symbols will be useful as subpictures of the final circuit. The simple circuits shown in Figure 17 were compounded into a big circuit involving about 40 transistors. Including much trial and error, the time taken by a new user (for the big circuit not shown) was ten hours. At the end of that time the circuit was still not complete in every detail and he decided it would be better to draw it by hand after all.

CONCLUSIONS

The circuit experience points out the most important fact about Sketchpad drawings. It is only worthwhile to make drawings on the computer if you get something more out of the drawing than just a drawing. In the repetitive

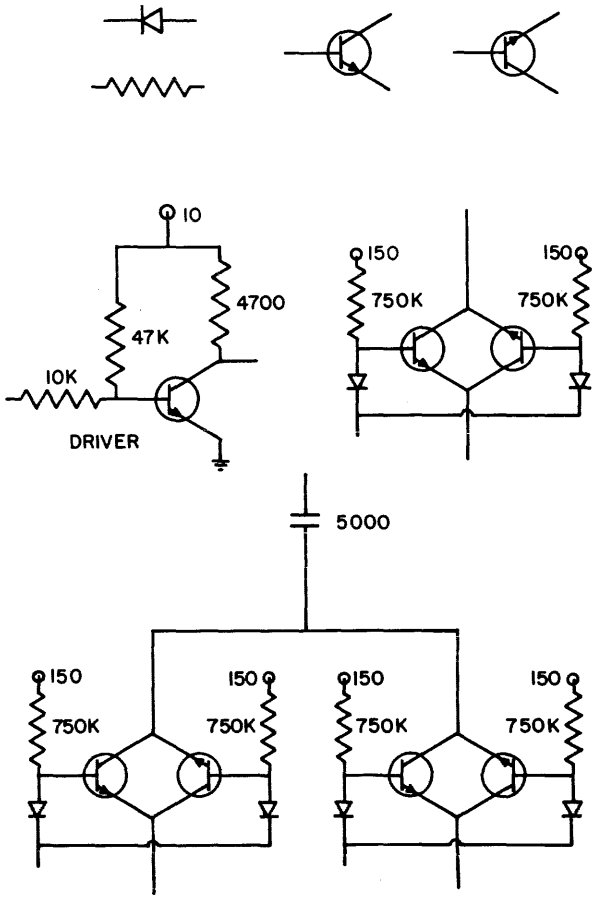


Figure 17. Circuit diagrams. These are parts of the large circuit mentioned in the text.

patterns we saw in the first examples, precision and ease of constructing great numbers of parts were valuable. In the linkage examples, we were able to gain an understanding of the behavior of a linkage as well as its appearance. In the bridge examples we got design answers which were worth far more than the computer time put into them. If we had had a circuit simulation program connected to Sketchpad so that we would have known whether the circuit we drew worked, it would have been worth our while to use the computer to draw it. We are as yet a long way from being able to produce routine drawings economically with the computer.

FUTURE WORK

The methods outlined in this paper generalize nicely to three dimensional drawing. In fact, the work reported in "Sketchpad III" by Timothy Johnson[3] will let the user communicate

solid objects to the computer. Johnson is completely bypassing the problem of converting several two dimensional drawings into a three dimensional shape. Drawing will be directly in three dimensions from the start. No two dimensional representation will ever be stored.

Work is also proceeding in direct conversion of photographs into line drawings. Roberts reports a computer program[9] able to recognize simple objects in photographs well enough to produce three dimensional line drawings for them. Roberts is storing his drawings in the ring structure described here so that his results will be compatible with the three dimensional version of Sketchpad.

Major improvements to Sketchpad of the same order and power as the existing definition copying capability can be foreseen. At present Sketchpad is able to add defined relationships to an existing object drawing. A method should be devised for defining and applying changes which involve removing some parts of the object drawing as well as adding new ones. Such a capability would permit one to define, for example, what rounding off a corner means. Then, one could round off any corner by pointing to it and applying the definition.

ACKNOWLEDGEMENTS

The author is indebted to Professors Claude E. Shannon, Marvin Minsky and Steven A. Coons of the Massachusetts Institute of Technology for their help and advice throughout the course of this research.

The author also wishes to thank Douglas T. Ross and Lawrence G. Roberts for their help and answers to his many questions.

BIBLIOGRAPHY

1. CLARK, W. A., FRANKOVICH, J. M., PETERSON, H. P., FORGIE, J. W., BEST, R. L., OLSEN, K. H., "The Lincoln TX-2 Computer," Technical Report 6M-4968, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., April 1, 1957, *Proceedings of the Western Joint Computer Conference,* Los Angeles, California, February 1957.
2. COONS, S. A., *Notes on Graphical Input Methods,* Memorandum 8436-M-17, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.
3. JOHNSON, T. E., "Sketchpad III, Three Dimensional Graphical Communication with a Digital Computer," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963, (this issue).
4. JOHNSTON, L. E., *A Graphical Input Device and Shape Description Interpretation Routines,* Memorandum to Prof. Mann, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.
5. LICKLIDER, J. C. R., "Man-Computer Symbiosis," *I.R.E. Trans. on Human Factors in Electronics,* vol. HFE, pp. 4-10, March 1960.
6. LICKLIDER, J. C. R., and CLARK, W., "On-line Man-Computer Communication," *Proceedings of the Spring Joint Computer Conference,* San Francisco, California, May 1-3, 1962, vol. 21, pp. 113-128.
7. LOOMIS, H. H. JR., Graphical Manipulation Techniques Using the Lincoln TX-2 Computer, Group Report 51G-0017, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., November 10, 1960.
8. MOORE, E. F., "On the Shortest Path Through a Maze," *Proceedings of the International Symposium on the Theory of Switching,* Harvard University, Harvard Annals, vol. 3, pp. 285-292, 1959.
9. ROBERTS, L. G., *Machine Perception of Three Dimensional Solids,* Ph.D. Thesis, Massachusetts Institute of Technology, Electrical Engineering Department, Cambridge, Mass., February 1963.
10. ROSS, D. T., RODRIGUEZ, J. E., "Theoretical Foundations for the Computer-Aided Design System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963, (this issue).
11. SOUTHWELL, R. V., *Relaxation Methods in Engineering Science,* Oxford University Press, 1940.

12. STOTZ, R., "Man-Machine Console Facilities for Computer-Aided Design," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963, (this issue).

13. VANDERBURGH, A. JR., *TX-2 Users Handbook,* Lincoln Manual No. 45, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., July 1961.

14. WALSH, J. F., and SMITH, A. F., "Computer Utilization," *Interim Engineering Report 6873-IR-10 and 11,* Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., pp. 57-70, November 30, 1959.

15. Handbook for Variplotter Models 205S and 205T, PACE, Electronic Associates Incorporated. Long Branch, New Jersey, June 15, 1959.

Sketchpad III

A COMPUTER PROGRAM FOR DRAWING IN THREE DIMENSIONS

Timothy E. Johnson
Mechanical Engineering Department
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

INTRODUCTION

An important area of investigation for the M.I.T. Computer-Aided Design Project[1] is the development of a facility for three-dimensional shape description. The heavy dependence of mechanical design upon three-dimensional objects makes such a facility an indispensable part of the full system which is envisioned. This paper describes the unique features required for three-dimensional graphics. Although the Sketchpad III System described here shares many features and routines with the Sketchpad System described in the preceding paper by Sutherland,[2] the extension of graphical techniques from two to three dimensions introduces many added requirements.

The design of a mechanical or structural object begins with a graphical description. Initially this description is not a precise statement of refined detail, but is a vague stirring of the imagination. This concept is slowly developed into the finished design through modification, deletion and analysis.

Thus, if a computer system is to work in partnership with a designer, the system must be able to accept, interpret, modify and remember shape description information introduced graphically. Graphical communication between man and machine need not be similar to pencil and paper methods. The digital computer permits the adoption of new graphical techniques, but these new methods must not be so far removed from "drawing board simplicity" so as to render the system inconvenient.

Shape description of a structural object must convey three-dimensional information to both man and computer. In modifying the part, the designer must be able to examine the structure from any attitude in space. Sections of the object must be easily oriented to permit meaningful graphical manipulations.

General three-dimensional graphical communication, which deals with arbitrary surfaces and space curve intersections, presents many difficult problems; the beginning has been modest and much work remains before the complete graphical communication problem is solved.

This work has been made possible through the support extended to the Massachusetts Institute of Technology, Electronic Systems Laboratory by the Manufacturing Technology Laboratory, ASD, Wright-Patterson Air Force Base under Contract No. AF-33(600)-42859. It is published for technical information only and does not necessarily represent the recommendations or conclusions of the sponsoring agency.

The TX-2 computation facilities of M.I.T. Lincoln Laboratory were made available to the project.

SCOPE OF PRESENT ACTIVITY

A prototype graphical communications system capable of manipulating straight line, "wire frame," figures in three-dimensional space is now in operation.* Neither program writing nor a knowledge of computers is required to operate the system. The definition, construction, and manipulation of three-dimensional surfaces are not included at present; hence edges which are normally hidden by forward surfaces are not obscured as they should be. Since all edges are visible, one views a "wire frame" with no covering.

MAN-COMPUTER INTERFACE

Real-time bilateral communication between man and computer is a prerequisite for a successful system. The CRT and light-pen (described below) meet the hardware input-output requirements of fast, two-way operation.

a. Output: Visual Presentation

Graphical images of three-dimensional objects are displayed on-line on a standard cathode ray tube. Because the screen is two-dimensional and the objects are three-dimensional wire frames, several viewing conventions were adopted to aid in visualizing the object in three-dimensional space. Stereo-optic displays and similar methods of creating space sensations were not considered because of clumsiness and because of bilateral communication problems.

Four views of the object are displayed by program, one in each quadrant of the CRT screen (Figure 1)*. A perspective view of the object appears in the upper right quadrant, and three orthogonal views in the remaining quadrants: top view—upper left, front view—lower left, and side view—lower right.

Wire frame objects displayed in a single two-dimensional view without perspective fail to convey depth information. Perspective gives the illusion of three dimensions by supplying the familiar convergence of lines as they recede from the viewer. A single perspective view of an unfamiliar object does not convey visually all the correct information either; for example,

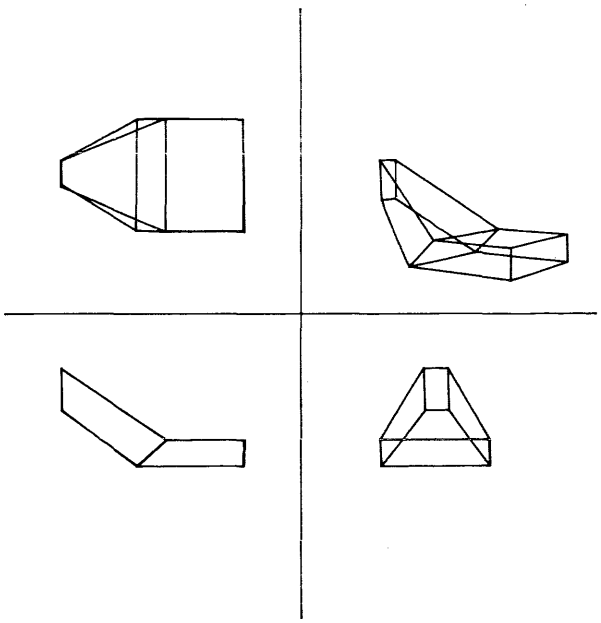


Figure 1. Typical graphical presentation showing top, front, and side views plus a "¾" perspective view.

are the receding lines parallel or do they *actually* converge? Hence at least one other complementary view is necessary. Three projectively related orthogonal views were chosen for the complementary function. There are several reasons for this choice: a) The three views completely describe a straight line object in three dimensions. b) Three ninety-degree rotations of the part are simultaneously in view, reinforcing depth perception. c) Many users of Sketchpad III are uncomfortable sketching in perspective and would prefer the orthogonal system used by most draftsmen.

b. Input: Sketching

A light-pen is used to guide drawing on the face of the CRT. The light-pen is a photo-diode mounted in a pen-like housing which is connected to the computer. A lens system in the pen housing focuses light on the photo diode giving a field of view of approximately one-half inch when the pen is held within three inches of the CRT screen. The pen, which acts only as a receiver, responds if a scope phosphor is intensified within its field of view and interrupts the computer momentarily; suitable programming determines which point caused the response.

* Programmed for the TX-2 Computer, M.I.T. Lincoln Laboratory.

* All figures in this paper describing 3-D objects were drawn with Sketchpad III.

This point is associated with the name of a line which is used as the entry point to the data structure. The line is either contained in the drawing or contained in a tracking cross. The tracking cross is used to determine the position of the pen on the scope face when the pen is not pointing at part of the drawing. Programming interprets the motions of the pen as the operator moves it across the screen. The light-pen permits the CRT screen to pass information in two directions; the CRT is simultaneously both an input and an output device.

Pushbuttons are provided which enable the operator to direct the computer program; for example, to erase what the light pen is pointing at, to move what the light pen is pointing at according to subsequent pen motions, to start drawing a straight line where the pen is pointing according to subsequent pen motions, to translate the drawing according to pen motions, and so forth.

The program interprets the rotations of three digital shaft encoders to mean: 1) magnify or reduce the drawing, 2) rotate the drawing clockwise or counterclockwise, 3) force or relax the perspective (by changing the convergence of the lines).

GRAPHICAL TRANSFORMATIONS

The four projections viewed on the scope are not four independent displays of stored two-dimensional information; rather, space coordinates in one data structure are transformed into two-dimensional images for display. Rotating, translating, magnifying, and changing the perspective does not affect the data structure (local transformations excluded). A line being drawn in any one view is simultaneously seen in the three other views; lines are drawn directly in three-dimensions and simultaneously fed back for display.

Rotation, magnification, translation, and perspective transformations are performed by a single 4 × 4 matrix.* Two matrices are used for display purposes: one for the perspective view and the other for the three orthogonal views—thereby enabling the perspective view to be manipulated independently of the orthogonal views.

* Developed by Larry E. Roberts at the Massachusetts Institute of Technology for use in his doctorate thesis on assembling three-dimensional descriptions of objects from their photographs.

The matrix can be viewed as a partitioned matrix of four parts (Figure 2). The upper left is a 3× 3 rotation matrix. The lower right position is the scale factor. The row at the bottom contains the three translation terms. The column on the right holds the inverses of the three viewing distances used in the perspective transformations.

The matrix operates on homogeneous coordinates (a, b, c, s). A three-dimensional point is determined by the ratio of its homogeneous coordinates:

$$a:b:c:s = x:y:z:1$$

In other words, the s coordinate is a scale factor and,

$$x = a/s$$
$$y = b/s$$
$$z = c/s$$

A typical transformation is shown in Figure 2.

The rotation section of the matrix is changed by post-multiplying by a second rotation matrix. This second 3 × 3 matrix describes relative two-dimensional rotations about one of the three orthogonal axes of the scope screen.

To cause a rotation, the operator first selects an axis of rotation by pointing at one of the three orthogonal views with the light-pen. Then the rotation shaft encoder is turned and the

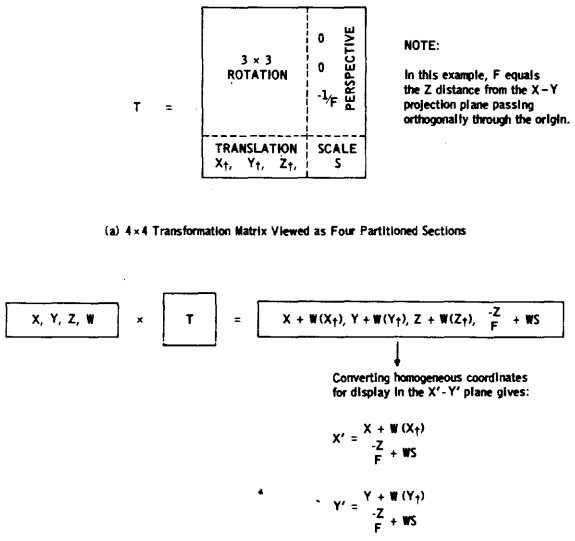


Figure 2. Matrix configuration used to perform multiple graphical transformations.

part rotates about an axis perpendicular to the selected viewing quadrant. The program calculates the sine and cosine of the shaft angle and positions the results in the second matrix according to the selection of axis. Post-multiplication takes place and the second matrix is cleared. The new transformation is applied to the drawing and a new display is "painted" on the scope screen. The program continually samples the knob position; as long as the operator continues to turn the encoder, the process repeats. The cycle is fast enough for simple drawings to give the illusion of a moving picture. The part moves relative to two of the three possible axes of rotation in three dimensions as the picture is rotated about one of the orthogonal scope axes (Figure 3). This method of rotating about one of the scope axes is visualized more easily than rotation about some axis fixed with respect to the part. Since the part can be rotated to any attitude in space by the twist of a knob, projectively related auxiliary views of the object can be generated at once.

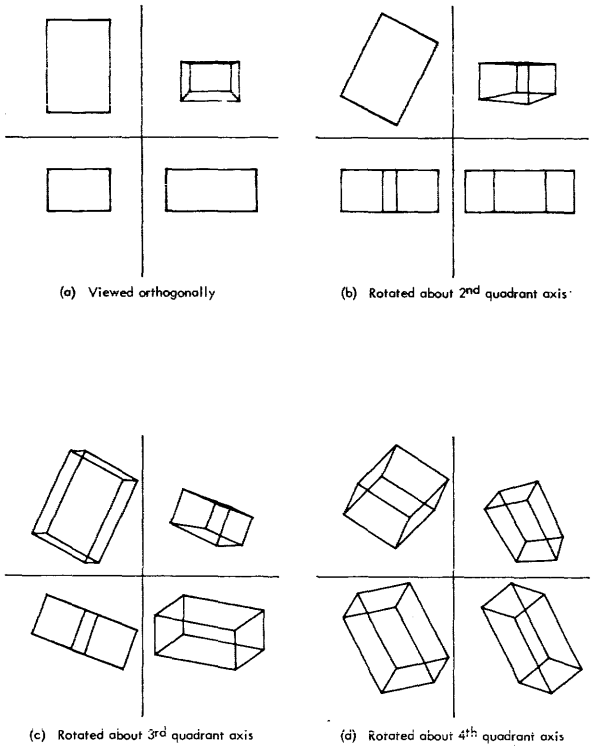


Figure 3. Additive rotation, beginning at (a), of a "wireframe" box about axes perpendicular to the orthogonal viewing quadrants. The 1st quadrant in each subfigure shows a perspective view of the 3rd quadrant, or front view.

The remaining sections of the transformation matrix are manipulated in similar fashion. Application of the matrix to segments of the object singled out by the light pen gives the important local transformation feature. Thus, graphical modification, which is so vital in the design process, is readily available.

DRAWING IN THREE-DIMENSIONS ON A TWO-DIMENSIONAL SURFACE

Moving a light pen across the face of a CRT and having the correct three-dimensional information pass into computer memory requires a simple method of specifying depth coordinates. Several methods were considered, such as assigning the depth coordinate to the pen location by positioning a shaft encoder or joystick. Approaches like this were dismissed because of the nearly "artistic" talent an operator had to have to visualize movements of the point.

The method finally adopted utilized the rotation facility. Because the part can be rotated to any position in space, lines can be drawn directly in three dimensions by drawing in a plane. The part is rotated until the area in which the line is to be drawn is parallel to a viewing quadrant. The line is then drawn true length; the depth coordinate remains constant as the pen moves across the plane of the scope screen. Specification of the single depth coordinate is done by program interpretation as described below. This program, called the Pen Space Location program is the backbone program of three-dimensional sketching. Sophisticated drawing is made possible by this program. Because of its generality, rotation of the part to bring sections into true view is not always necessary.

PEN SPACE LOCATION PROGRAM

The Pen Space Location Program performs two important functions: a) It defines a point in space called the *P*en *S*pace *L*ocation (PSL), by assigning a depth coordinate to the pen location according to the pen's previous position in another view, and b) it permits precise positioning of the PSL with coarse light-pen motion. It is the motion of the PSL that guides the drawing of lines.

a. *Arbitrary Depth Assignment*

The three orthogonal views represent three adjacent faces of an imaginary cube surrounding a part in space; the edges of the part are projected onto each face. A point on any face defines a projection line in space perpendicular to the selected face.

When defining a three-dimensional point, the Pen Space Location program begins the definition by interpreting the projection line of a two dimensional point positioned by the pen as the intersection of two temporary, invisible planes parallel to the faces of the imaginary cube. To complete the definition of the point in space, a second point is positioned in either of the other two views and its projection line will then intersect one of the two temporary planes (Figure 4). This intersection defines the point in space.

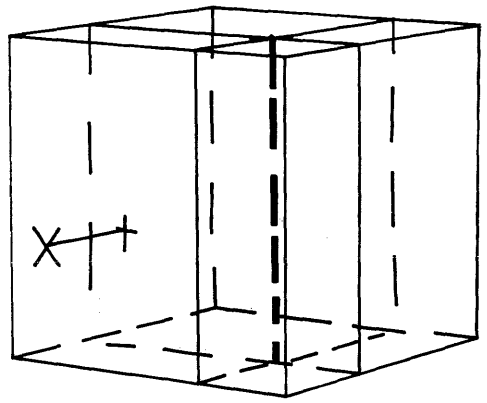


Figure 4. Graphical interpretation of method used in the PSL program to define a 3-D point. The cube represents the imaginary volume enclosed by the orthogonal viewing quadrants. In this example, the *heavy* dashed line indicates the projection of the first point positioned in the top view. (Dashed lines are used for visualization purposes only.) This line is interpreted as the intersection of two planes. The small '*X*' represents the second point positioned in another view. The solid line radiating from the '*X*' is its projection. The '+' defines the 3-D point.

The two-dimensional position of the light-pen on the scope face is established by a tracking program. This program calculates the coordinates of the center of the pen's field of view and remembers the pen's position when the pen is removed from the scope face.

When the pen begins tracking in a second view, the Pen Space Location program obtains the appropriate coordinate from the remembered point of the first view for use as the depth coordinate of the current pen position. This is the pen position in space called the Pen Space Location and its projections are displayed as bright dots in all the viewing quadrants. As the pen is moved in one view, the depth coordinate of the PSL projected in front of the pen remains fixed.

Lines are drawn by locating the PSL at an initial position, depressing a Start Draw button, and moving the pen. As long as the pen tracks, a line appears between the initial point and the PSL. Removing the pen from the screen with a flick of the wrist terminates motion at the last position of the PSL.

b. *Precision Point Assignment*

To begin a line on an existing line, the PSL must be precisely positioned on the desired section of line. Since the light pen motions directing the movement of the PSL through space are quite coarse, programming must direct the precision movements of the PSL.

The PSL is surrounded by an imaginary sphere of one-eighth inch radius. The coordinates of the PSL and the radius of the sphere are converted to homogenous coordinates and transformed by the inverse of the 4×4 transformation matrix. When the pen is held over a line, the PSL program determines which line (or lines) is responsible for the light pen response and compares the stored line(s) with the transformed PSL. If a line passes through the sphere surrounding the PSL, the program calculates the point on the line nearest the center of the sphere and assigns the PSL this new value. As soon as the line passes outside this sphere (centered in front of the pen), the PSL assumes a value that is projectively related to the center of the pen's field of view. The new PSL retains the depth coordinate of the point on the line. The Pen Space Location program also determines end points and intersections of lines in this manner. Thus a steady hand is not required to construct accurate drawings.

The PSL program has a second mode that can be selected by push button. Instead of surrounding the PSL with a sphere, the projection of the PSL is surrounded by a cylinder. Whenever a line intersects the imaginary cylinder, the point on the line nearest the axis of the cylinder replaces the PSL. As in the other mode, when the

lines fall outside the detection volume, the new PSL retains the depth coordinate of the point on the line, and is projected in front of the pen. The cylinder mode is generally used: a) for selection of existing depth coordinates in one view. That is, it is not necesary to move the pen momentarily to a second view to establish a depth and drawing can proceed at a faster speed. This is particularly useful when drawing in the perspective view; b) to determine the location of a projected line in three dimensions. The PSL is "locked" on the line by moving the pen over the line in one view and the projection of the PSL can be observed in the other views.

The section of the PSL program that performs nearest point calculations is flow diagrammed in Figure 5. The cylinder mode is merely the special case where the depth coordinate is ignored. The mode selection button modifies the program so the proper third coordinate is ignored.

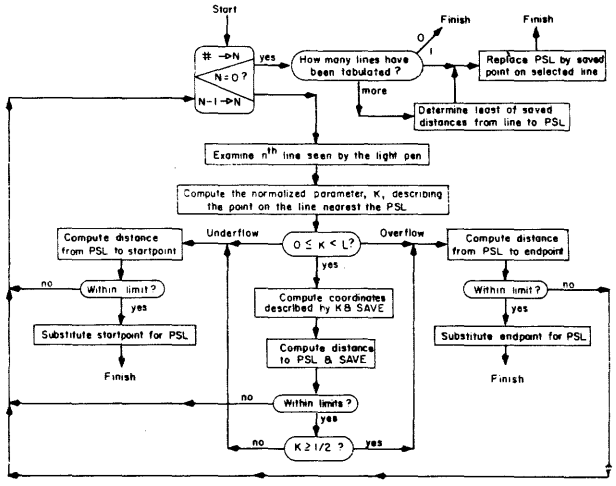


Figure 5. Flow diagram of the precision point assignment section of the PSL program.

SUPERPOSITION

The cylinder mode fails when lines are superimposed. Attempting to "latch" the light pen onto a line projected on top of another line confuses the program and the nearest point calculations are performed on the first line "seen" by the light pen. If a cube were rotated so it appeared as a square for example, the operator could not predict what depth the PSL would assume when the pen is held over a line. Thus, when the superposition occurs, the sphere mode must be used.

SYNOPSIS OF SKETCHPAD III PROGRAMMING

The data structure and the utility programs that generate the data structure used in Sketchpad III were developed by Ivan E. Sutherland at Massachusetts Institute of Technology for use in his two-dimensional graphical communications program, Sketchpad. A general introduction to the utility programs is given below, but the reader is referred to Reference 3 for complete details.

Graphical information is stored in an n-component list structure. The pointers connecting the n-component elements form closed rings and enable tracing of information in either direction. Different rings thread through several levels in an n-component element providing several paths to the same information. Each type of information, such as a line or a point, is referenced by a single block. These blocks contain pointers to the appropriate transformation and display subroutines. Each of these distribution blocks in turn are grouped according to generic type.

The program that operates on the structure is generalized. An executive routine determines what *type* of graphical manipulation must be performed by periodically sampling knobs and push buttons. The graphical information to be manipulated, indicated by the light pen or push buttons, is located in the list structure; the program subsequently transfers to its distribution block via the list structure. The distribution block in turn transfers control to the subroutine which determines *how* the manipulation is done. Thus, the program is composed of many modules which are interconnected by the list structure itself.

Many features have been adapted from Mr. Sutherland's program to increase the flexibility of Sketchpad III. These include:

1. Storing several pictures in computer memory, subject to immediate recall.
2. Storing pictures on magnetic tape.*
3. Merging or combining end points of lines.
4. Merging several lines into one line.
5. Generating hard copy on an X-Y plotter.*

* Program by Leonard M. Hantman of Lincoln Laboratory.

** Ibid.

A program, now almost completed, inserts one picture into another picture. This feature accelerates the drawing of repetitive structures and the integrating of components into assemblies.

FUTURE ACTIVITIES

Many difficult problems must be surmounted before a general graphical system is operating. Methods fast enough to satisfy real-time requirements must be devised that:

1. Define arbitrary surfaces.
2. Determine space-curve intersections of two surfaces.
3. Determine edges hidden by arbitrary surfaces.
4. Satisfy general graphical constraints.

As the system evolves, computers will be applied throughout the design-to-manufacturing spectrum. Design analysis (stress, dynamic, etc.) capabilities will be embraced by the system and operate directly upon and influence the stored graphical information. Manual part programming for numerically controlled production machines can be by-passed. The goal is to decrease the time spent preparing a part for production (lay-out, detail, drafting, etc.) from months to days.

BIBLIOGRAPHY

1. COONS, S. A., "An Outline of the Requirements for a Computer-Aided Design System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume.)
2. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Communication System," *Proceedings of the Spring Joint Computer Conference,* Detroit, Michigan, May 21-23, 1963. (This Volume.)
3. SUTHERLAND, I. E., "Sketchpad, A Man-Machine Graphical Communication System," Ph.D. Thesis, Massachusetts Institute of Technology, Electrical Engineering Department, Cambridge, Massachusetts, January 1963. (To be issued as a MIT Lincoln Laboratory Report.)
4. COONS, S. A., "Notes on Graphical Input Methods," Memorandum 8436-M-17, Dynamic and Control Laboratory, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Massachusetts, May 4, 1960.
5. JOHNSON, T. E., "Sketchpad III, Three-Dimensional Graphical Communication with a Digital Computer," S.M. Thesis, Massachusetts Institute of Technology, Mechanical Engineering Department, Cambridge, Massachusetts, June 1963. (To be issued as a MIT Electronic Systems Laboratory Report.)

KEY ADDRESSING OF RANDOM ACCESS MEMORIES BY RADIX TRANSFORMATION

Andrew D. Lin
Information Storage Systems
IBM General Products Division
Development Laboratory
San Jose, California

INTRODUCTION

The addressing procedure for nonsequentially stored data in large capacity random access memories is the automatic allocation of storage position to each member of a data set. Each member, termed here as a record, is identified by a unique description or key which is part of the record. Key addressing then is the execution of a mapping function which mathematically relates the key of a record to its location in storage. To store or to retrieve a record, its key is presented to the addressing facility which then generates an address to which the memory will then access.

Various addressing techniques have been used in the past and are being used at present. Typically, however, they have been individually tailored to the application. The growing usage of large capacity random access memories is placing practical urgency on a unified approach to the addressing procedure. A unified approach implies freedom from the parameters, to be discussed later, which have governed the design of past techniques. A further consideration would recognize that random access bulk storages will increasingly appear as the central machine element in information storage systems in contrast to its peripheral role in present data processing systems. This then calls for a unified solution which lends itself simply and economically to hardware implementation yet permits its execution by stored program. The subject investigation was oriented toward that objective.

The Storage Address Set

The storage devices within the purview of this paper are large capacity random access bulk memories. In general, there is no difference in the addressing problem of a memory which depends on mechanical means for accessing and of a memory which depends on electronic means for accessing. In practice, however, mechanical access time is several orders of magnitude greater than electronic access time; hence, it becomes more significant in the former to reduce the number of seek cycles of the actuator for the average record in the data set.

These memories are usually organized in a binary or decimal hierarchy to agree with the radix of the arithmetic unit of the processor of whose system the memory is destined to be a member. The storage is generally divided into q^n subspaces where q is the number base of the organization structure and n is some integer. The latter, of course, corresponds to the number of positions in the address register.

A subspace or group of subspaces is termed a bucket if, once the access mechanism has arrived, its entire contents may be scanned without further change in the contents of the

address register. For mechanically accessed memories, buckets tend to contain a plurality of records. A good reason for this is that the development of higher bit density in magnetic recording is advancing more rapidly than that of improved resolution and speed of access mechanisms. Hence, the tendency is for bucket capacities to increase in the future since a larger share of the burden of search will have to be shifted to the faster scan-time capability within the bucket and away from the slower access mechanism. This shift is necessary to strike a reasonable balance between access time and scan time.

Random access memories of the current generation are location-addressed rather than content-addressed. A bridging function is therefore necessary to associate the keys of a data set with memory locations. If, in the rare case, the key happens to be directly usable as an address or if the predetermined address is appended to the key, then "direct addressing" is possible. If the association is by means of arbitrary assignment of locations to keys, the process is called "table look-up". Table look-up may consist of one or more stages with the stages stored either in internal memory or in the bulk storage or a combination of both. In "key addressing", the key is operated on by a mathematical routine or algorithm to generate an address. In effect, then, key addressing makes the random access memory appear as if it were content-addressed, based on a record's unique descriptor.

A machine address set has these stable characteristics: It is numeric, a solid integer set in its range; consecutively ordered, and does not change with time. These are in sharp contrast to those of a data key set, which will be discussed next.

The Data Key Set

The data key set in typical random access memory applications has the following characteristics. The length of keys, of course, varies from data set to data set, with the maximum ranging up to the neighborhood of 20 characters. Members of a given key set frequently differ in length, with blanks often appearing in various positions of the allotted field. Character positions may be differently restricted as to allowable symbols such as numeric, alphabetic, alphameric and special signs; i.e., keys may very well be mixed radix representations.

A data key set is typically a small fraction of the total allowable combinations or parent set which the key length and the source alphabet permit. If the parent set were arranged in collating sequence, it may be seen that the keys, as a subset, generally populate the range in an irregular manner, giving the impression of uneven clusters separated by huge gaps. In part numbers, for example, this is due to the classification logic adopted by the user; in English names, on the other hand, the phonic requirements and the relative incidence of vowels and consonants contribute to clustering in the spectrum of all possible combinations.

To a user, then, which specific subset in the parent set presents itself as the key set is probabilistic. Furthermore, in a dynamic application, the cluster and gap configuration will change with time because of accretions and deletions in the data set. In view of the above, probabilistic techniques must be used to provide the association or storage mapping function of keys to addresses.

Occupancy Distribution Considerations

In key addressing, the mapping procedure must distribute the keys of a set over the memory bucket addresses as evenly as possible. Ideally, the user desires, if the need arises, to load his memory to 100% of capacity without special fitting operations requiring complex programming; he also wants to find a record in exactly one access cycle. In other words, he would like to find his record always at the address which the mapping algorithm generated, i.e., in its home bucket. Both requirements imply a perfectly even distribution, which a probabilistic technique on a nondeterministic key set can only approach. The proper performance objective, therefore, is to achieve a mapping that is equivalent to what a randomly distributed set of elements would experience if mapped into a set of buckets, each having equal probability of being occupied. The expectation then would be a Poisson distribution.[1] Clearly, the addressing algorithm sought should provide the conditions which would permit a similar expectancy. In short, the criterion of mapping performance should be the Poisson distribution.

Bucket Overflow Considerations

A key addressing technique involves two stages: first, a mapping procedure to allocate the records as evenly as possible among the buckets; second, an overflow procedure to divert the overflow into trailer buckets. Once the occupancy distribution is made, the bucket capacity, C, determines the extent of the overflows. Obviously, the poorer the mapping performance, the greater the burden left for the overflow routine. The effectiveness of the latter lies in its added levelling effect on the distribution. This implies, for the average record in the data set, a reduced seek factor, S; i.e., the number of seek cycles of the access mechanism to trace it to its overflow location. The criterion of merit for the composite procedure of mapping and overflow disposition is most usefully expressed as the percentage, R, of the data set that is lodged into home buckets. The complement of R, of course, is the overflow percentage, F, that finds placement in trailer buckets. It might be added here that in applications where a high percentage of activity is confined to a small portion of the data set, the overflow problem becomes almost insignificant when the most active records are loaded into their home buckets and the less active ones are loaded into trailer buckets.

Parameter Relationships

The qualitative relationship among the parameters in an addressing technique are as follows: Consider any data set of N records occupying a fraction L (load factor) of the total space in memory of K buckets, each having a capacity of C records. For any memory of given K and C, the seek factor improves (i.e., decreases) with a drop in the load factor. This may be seen in a preview of Figure 5. For a memory having product KC constant, but with K and C inversely variable, the seek factor improves with higher bucket capacity. A glance at Figure 4 also indicates the relationship between the load factor L, the ratio of data records to buckets $N/K = \overline{b}$, and R, the percent of the data set lodged into home buckets. For a given L, R improves with a higher value of $\overline{b}$. These are relations to be borne in mind in subsequent pages of this paper.

Source Alphabet Considerations

A key is converted from its human-readable form to its coded representation before it can be machine-processed by an address transformation technique. This processing is arithmetic and, therefore, must deal with numeric digits. With nonnumeric characters in a key, the user faces the alternative of recognizing only the numeric component of the coded representation (for example, in the Binary-Coded Decimal code, stripping the two zone bits from the 6-bit configuration) or of devising some method of having all bits participate in the calculation process. If, as has been done frequently in past techniques, the nonnumeric bits are ignored, then the effect is to impose on the human-readable key set a condition where certain bit-coded subsequences have a one-to-many correspondence with human language characters in the key. This, of course, intensifies the clustering effect already present in the human readable key set. Clearly, this should be avoided in any addressing algorithm.

System Considerations

In planning an application for a random access bulk storage, the user has a certain latitude in assigning values to the critical parameters. If mapping performance were predictable in relation to these parameters, planning would become more systematic. For example, a user having chosen a certain load factor, L, and a certain ratio of records to buckets, b, could with confidence estimate the approximate percentage of the data set that would overflow. The mapping algorithm should, therefore, be reasonably consistent and predictable in performance for a given set of parameters.

Again, from the user's viewpoint, it is highly desirable that an addressing technique be directly applicable to any key set as found. In other words, the user would be glad to be freed of the task of preanalyzing and experimenting with the structure of a new key set to discover the series of procedures which will "randomize" it. He would prefer a general-purpose scheme which he can immediately apply to his key set with the high expectation of getting a transformed, randomly distributed set.

The primary interest in this paper on hardware embodiment of the addressing algorithm is based on the following system considerations. As a self-contained addressing facility, it can be integrated into a variety of system environments, both processor-centered and storage-centered. In the former, it would serve to over-

lap the addressing procedure with the main processing program. Additionally, it may serve as a time-shared device placed between one or more processors and an array of bulk storages (having same or different parameters). If additionally the hardware embodiment of the addressing algorithm should permit transformation of the key optionally into binary or decimal addresses, a further system advantage is gained. This would relieve bulk storages of the design restriction that its address radix be necessarily the same as the computing radix of the processor.

Storage-centered systems will, in general, not be equipped with as high a level of arithmetic and logical capability as processor-centered systems. A separate addressing facility would make it possible for them to become truly autonomous systems, not necessitating the association of a general-purpose processor to enable use of bulk storage for strictly retrieval purposes.

Summary of Requirements

To the above considerations, others may be added to compile a list of functional objectives. These are proposed here as the basis for any unified solution to the addressing problem.

1. The addressing algorithm should distribute the records among the memory spaces as evenly as is probabilistically possible, with the Poisson distribution as the objective.
2. The composite procedure of mapping and overflow tracing to store or retrieve the average record should be fast in terms of a low number of seek cycles.
3. The algorithm should be universal in application for all keysets. It should be independent of source alphabet, machine character code and key length.
4. The algorithm should have the element of predictability in the sense that a prospective user, electing specific values for some of the principal parameters (load factor and keys-to-buckets ratio) can have high expectation of a certain mapping performance, i.e., the percentage of the data set that will overflow.
5. The algorithm should be directly applicable to any key set, as found; i.e., without pre-analysis or pre-editing.
6. The algorithm should be implementable as independent hardware simply and economically, preferably using a systematic sequence of elementary operations. Time of execution should be a relatively low multiple of the data transfer rate of the using system.
7. Hardware embodiment of the algorithm should permit the option of decimal or binary output addresses.

Proposed Solution

The solution offered here for key addressing is briefly as follows. The key is introduced to the mapping process as a binary sequence, is interpreted as 4-bit p digits and radix-transformed into a magnitude expressed in address radix q, where p is relatively prime to q. Truncation yields the memory address. The rationale is given below.

As a first step in meeting the solution requirements, a common basis of representing all keys must be provided so that they may appear as a standardized input to the address-generating process. It has already been noted that human and machine language formats contribute to structuring in the key set. Eliminating the character partitions and expressing the coded form of a key as a binary sequence, i.e., a continuous train of ones and zeros, would reflect this compound structuring. The key representation is now an amorphous binary array without the attribute of magnitude. It awaits a choice of radix and of grouping to quantify it. The rules of quantification and the subsequent processing should transform the original key set into a randomly distributed set of entities. They should also permit a systematic iteration of elementary operations to permit economical hardware implementation as well as stored program execution.

A grouping of 4 bits was chosen as the unit of processing; i.e., as the byte size. This is reasonable since addresses are generally numeric and hence require 4 bits for a decimal digit. At the same time 4 bits is compatible with a binary address radix, since a magnitude expressed in a radix that is an integral power of 2 can always be read out as a pure binary number.

The results of processing successive bytes should be cumulative in the hardware implementation. Compression to the range of the addresses should be progressively done, pref-

erably by the simple operation of truncation by the physical bounds of the device itself.

To evolve our transformation algorithm, we start with the hypothesis that the principal phenomenon of key sets is the tendency to cluster in various ways and that this is the only significant reason which causes key sets to deviate from randomness.

We observe that a sequence of keys may occur for which all but a few bits of its coded representations remain constant. Such a group we call a cluster. The allowable range of a key set may be thought of as an n-dimensional hypercube whose 2^n vertices represent permissible n-bit keys. A typical key set is some subset of these vertices. As a simplified example, a vertex in a 5-dimensional space may be written as 10111. An immediately adjacent vertex would differ in only one bit position and we say it is one Hamming distance[4] away. For example, 10101 is adjacent to 10111. The vertex 11101 would be two distances away from 10111, and so forth. A cluster in this concept would be a collection of vertices relatively close to each other. Keys belonging to such a cluster are identical in most of their bit positions while the rest are variable. The variable bits are, of course, not necessarily in adjacent positions in the key nor at the low-order end. Two examples of clusters in human readable language are shown in Figure 1.

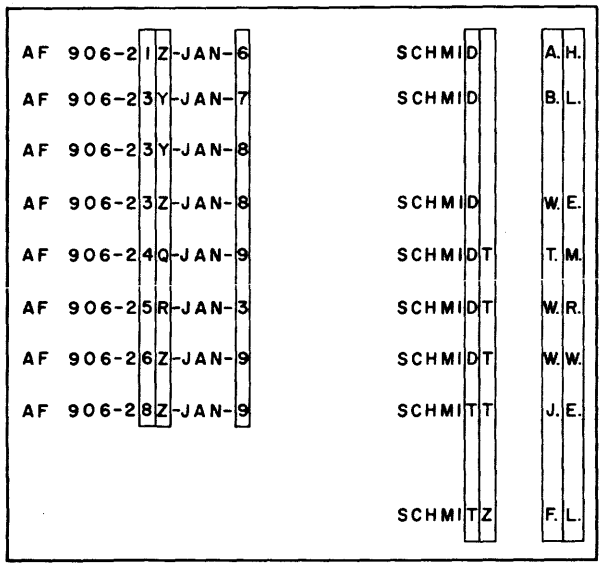


Figure 1. Examples of Clusters.

We propose the criterion that a good address mapping must destroy clusters. This is to say that clustermates (members of the same cluster) should tend not to be bucketmates (members of the same bucket). In other words, bucketmates should not have their inverse images in the same cluster. We now offer an address mapping using radix transformation to destroy these clusters and to satisfy the solution requirements previously mentioned. The theoretical justification is developed below.

Suppose that a given memory is addressed with a radix q. Usually q will be either 2 or 10. The number of buckets, K, is equal to q^m where m is most naturally an integer. The amorphous binary array is now treated as a number expressed in binary-coded p-radix digits where p is prime to q. Radix transformation is then performed on the key to convert it from base p to base q using the familiar formula:

$$\sum_{i=0}^{n-1} d_i p^i$$

where n = the number of p-radix digits in the key array

$$d_i = \text{value of } i^{th} \text{ digit}$$

The result is a randomly distributed set of transformations in q-radix representation, which upon truncation by modulus K, yields the desired addresses A; i.e.:

$$A = \sum_{i=0}^{n-1} d_i p^i \bmod K$$

For example, let $p = 11$, $q = 10$, $m = 4$, the key be Smith and the machine language be binary-coded decimal (bcd). Suppose 3 bits at a time are grouped to form 11-ary digits. Then the steps *a*, *b*, *c*, *d*, would ensue.

	S	*M*	*I*	*T*	*H*
a.	010010	100100	111001	010011	111000
b.	010 010	100 100	111 001	010 011	111 000
	2 2	4 4	7 1	2 3	7 0

c. $(2,244,712,370)_{11} = (5,230,793,172)_{10}$

d. Address = $5,230,793,172 \bmod 10^4$
= 3172

When will any two keys X and Y map into the same bucket? Only when they are congruent q^m. We may express this as

$$X \equiv Y \bmod q^m$$

which implies $X = Y \pm a\ q^m$ for some positive integer, a. If this is true when X, Y, and q^m are simply magnitudes without radix representation, then the equation holds also for any radix subsequently chosen to express it.

For example, let:

$X = |123|$, $Y = |23|$, $a\ q^m = |100|$,
where $||$ means the *magnitude*, regardless of what radix is used to express it.

Then:

$$|123| = |23| + |100|$$
$$(123)_{10} = (23)_{10} + (100)_{10}$$
$$(146)_9 = (25)_9 + (121)_9$$

However, consider the difference when X and Y are considered initially as amorphous binary arrays which have no attribute of measure until quantified by choosing a radix p as basis for interpretation. Note now that two arrays quantified in p-radix, $(X)_p$ and $(Y)_p$ (bucketmates), which differ by a magnitude aq^m are not usually congruent when evaluated in a radix p, where $p \neq q$. For example, if two arrays are 123 and 023, it can be seen that:

Quantified in radix 10: $(123)_{10} \equiv (23)_{10} \bmod |100|$
Quantified in radix 9: $(123)_9 \not\equiv (23)_9 \bmod |100|$

We choose radix p relatively prime to q because aq^m, when expressed in p, has a high percentage of nonzero, non $(p-1)$ digits. (In contrast, when aq^m is expressed in q-radix, its representation is (a) followed by all (m) zeros.) This indicates that bucketmates expressed in base p will differ in many digital positions, which implies that they will differ in many bit positions. In the n-dimensional space concept, it follows that the inverse images of bucketmates in the key set are relatively distant from each other and hence do not come from the same cluster.

To make this argument more precise, consider the digital configuration of two keys that are made bucketmates by this method. Let x_i be the i^{th} digit of X, y_i the i^{th} digit of Y, and z_i the i^{th} digit of the magnitude aq^m expressed in p-radix. Then, from the addition:

$$\begin{array}{r} y_j\ y_{j-1} \ldots y_i \ldots y_3\ y_2\ y_1 \\ +\ z_j\ z_{j-1} \ldots z_i \ldots z_3\ z_2\ z_1 \\ \hline = x_j\ x_{j-1} \ldots x_i \ldots x_3\ x_2\ x_1 \end{array}$$

What are the conditions under which x_i and y_i differ? In general, they are different if z_i is nonzero or non $(p-1)$. In particular, lowest order x_1 and y_1 would be different for all nonzero values of z_1 since no carry is possible from the right. It may be noted further that, if value $(p-1)$ can never occur in x_i or y_i, an added condition is assured when z_i and z_{i-1} are consecutively 0 and $p-1$, or $p-1$ and 0. This suggests then that in interpreting the binary array of the key, the relationship of p to q may advantageously be $p = q + 1$. It will be seen later how this relationship is also of benefit in implementation. Examples of p-q combinations are 3-2, 5-4, 9-8, 11-10, 17-16, and so forth.

To illustrate that aq^m in p-radix does indeed have a high percentage of nonzero, non $(p-1)$ digits, an investigation was made of the numbers $a2^{10}$ expressed in base 3. For each value of (a), a function $d(a)$ was computed such that $X = a2^{10} + Y$. Then X differs from Y in at least $d(a)$ digital positions in their radix-3 representation. A frequency tabulation of this function for $a = 1, 2 \ldots 10{,}000$ is shown in Table 1.

Table 1—Frequency Distribution of Minimal Distances for $a2^{10}$ Expressed in Radix 3.

$d(a)$	Freq.	Cum. Freq.
3	7	100.00%
4	6	99.93
5	197	99.87
6	179	97.90
7	1046	96.11
8	782	85.65
9	2573	77.83
10	1503	57.10
11	2146	37.07
12	911	15.61
13	510	6.50
14	123	1.40
15	17	0.17
	10000	

This table shows that the radix transformation at least does not map into the same bucket two keys differing only in two bit positions; i.e., having a "distance" of two. In general, however, pair-wise relationships of this sort generally result in stronger affirmation of our thesis for larger key sets, each pair of the set having this relationship. For example, a set of three bucketmates, each pair of which differs

in at least two bit-positions, cannot as a set vary only in two bit-positions.

Table 1 assumes that (a) takes on the values 1, 2 . . . 10,000 with equal likelihood. It is not presumed that every subtle condition which contributes to x_i and y_i being different has been reflected in the tabulation. The table, therefore, confirms conservatively but strongly the cluster-destroying capabilities of this method of interpreting a key in p-radix and of transforming it from base p to base q to generate an address for a q-radix memory.

Hardware Implementation

To incorporate the proposed algorithm into hardware, one might elect a value for the p-radix which makes the hardware requirements simple and light. The q-radix, obviously, will either be 2 or 10, since random access bulk memories are, and most likely will be, addressed in either pure binary or binary-coded-decimal.

It has been noted why p must be relatively prime to q. Additionally, if p is chosen equal to $q + 1$, then conversion from p to q may be performed by the elementary operations of shifting and adding. For example, given a q-radix address register and a p-digit $(d)_p$ to be converted into base q, we would multiply d by $q + 1$. We would implement this in a shift register by adding d to one register position, then shifting (in effect, multiplying by q) to the next higher-order position and adding d again.

We wish to standardize the input so that a high proportion of common hardware may be used whether the device is to be designed for binary address output, for decimal address output, or for both. Partitioning the binary array of the key into 4-bit bytes would be suitable for reasons already explained.

Figure 2 shows a device which would implement the above. It consists of a circulating shift register of m digital positions $D_m \ldots D_1$, a single-digit load register D_0, a single-digit adder H, and associated logic. The incoming key enters as a serialized train of standardized 4-bit bytes. A byte at a time is loaded into D_0. Between bytes, the following occurs: The contents of D_1 and D_0 are added; the sum is circulated back to D_0, and if necessary, the carry trigger is set. A right shift follows and another add-shift cycle ensues. This cycle is performed

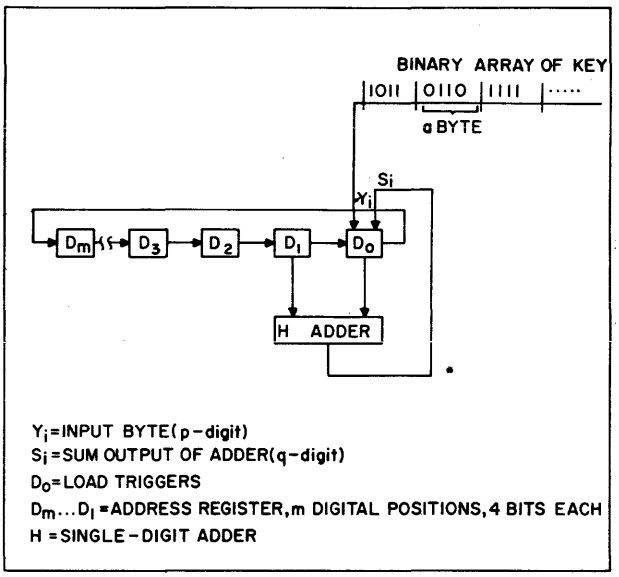


Figure 2. Universal Address—Generating Device.

m times for each byte. The next byte then enters D_0 and the sequence of operations is repeated. The process ceases when the key array is exhausted.

The finite length, m, of the register automatically truncates the cumulative sum of products at the conclusion of the processing of each byte. The final residue is the desired address. Truncation is valid since the following relation holds:

$$[\{a \bmod K + b\} \bmod K + c] \bmod K = (a + b + c) \bmod K$$

where a, b, c typically are successive bytes and $K = q^m$. It is apparent that the conversion to any address A is expressible as:

$$A = \sum_{i=0}^{n-1} d_i p^i \bmod K$$

where, for a binary array of n bytes, i equals 0, 1, 2, . . . $(n-1)$; and d is the digital value of the byte.

In the device, the above formula (less mod K) is advantageously restated in iterative form and implemented as follows:

$$A = p[p\{p[p\{\ldots p(\{p(d_n)\} + d_{n-1}) \ldots\} + d_3] + d_2\} + d_1] + d_0$$

This nesting arrangement suggests that the bytes, d, be processed in decreasing order of significance as each byte comes in from the binary array. To generate the address, the formula is evaluated progressively by working

from the innermost nest towards the outermost nest. Within each nest, the following occurs: a byte d_i is brought into D_0 and added to the product of the radix p and the current contents of the address register. Since $p = q + 1$, this product may be generated digit-by-digit by adding the current contents of D_1 into both the present and the next higher-order positions. Thus, the contents of D_1 participate in two successive add cycles: first when it is in D_1 and next when it is shifted to D_0. For example, in an 11-to-10•conversion, if the input byte is 3 and the current contents of the address register are 024, this happens:

$$\begin{array}{r} 3 \\ 44 \\ 22 \\ \hline 267 \end{array}$$

For choice of output addresses in either binary or decimal form, logic may be provided as shown in Figure 3. The binary form is, of course, quite straightforward. However, the decimal form requires that the input byte, if it exceeds 10 in value, be decomposed into a carry and a modulo 11 digit. Furthermore, the influence of the carry must be sustained for the first two add/shift cycles to correct the cumulative results in the address register. For brevity, the detailed logic will not be shown.

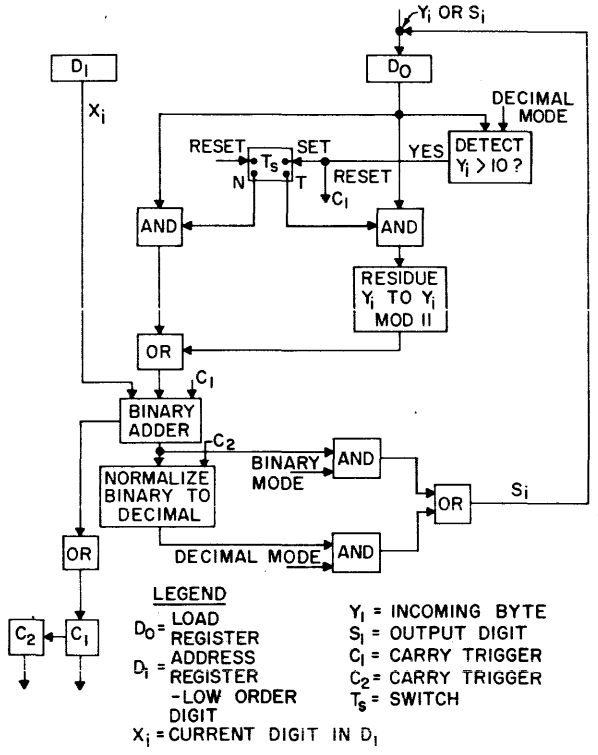


Figure 3. Block Diagram for Binary/Decimal Address-Generating Device.

An example is presented in Table 2 to illustrate how the binary array of a key having successive byte values of 1, 7, 1, 15 is converted from 11-radix to 10-radix representation; i.e., from $(1724)_{11}$ to $(2204)_{10}$.

It may be noted that if the incoming byte rate of the using system is J, the transformation rate per byte would be $\frac{J}{(2m+1)r}$ where r is the ratio of the speed of the circuit family employed in the address transformation device and the speed of the circuit family in the using system.

Empirical Verification

An extensive empirical testing program was performed on a IBM 704 computer to test the cluster hypothesis and the mapping performance of the radix transformation algorithm. Seven customer key sets were used. In addition, three generated sets of random alphameric patterns were processed. Table 3 shows their characteristics. Finally, a set of values were computed from the theoretical Poisson distribution to serve as criteria of performance. The several test objectives and their results are as follows:

The following notations will be used:

- k — number of binary positions in memory address register
- K — number of buckets in the memory $= 2^k$
- C — bucket capacity in records
- N — number of keys in key set
- b — bucket occupancy level (stochastic variable in frequency distribution)

$\bar{b}$—average number of records per bucket $= \frac{N}{K}$

M—number of record spaces in memory

L—load factor $= \frac{N}{M} = \frac{N}{KC} = \frac{\bar{b}}{C}$

R—percent of N lodged in home buckets

e_i—underflow or overflow in i^{th} bucket

g_i—net undisposed overflow at i^{th} bucket

f—number of buckets having occupancy b

TABLE 2

Example of 11-to-10 Conversion, Key to Decimal Address

D_4	D_3	D_2	D_1	D_0		C_2	C_1
0	0	0	0	1	I	0	0
0	0	0	0	1	A	0	0
1	0	0	0	0	S	0	0
1	0	0	0	0	A	0	0
0	1	0	0	0	S	0	0
0	1	0	0	0	A	0	0
0	0	1	0	0	S	0	0
0	0	1	0	0	A	0	0
0	0	0	1	0	S	0	0
0	0	0	1	7	I	0	0
0	0	0	1	8	A	0	0
8	0	0	0	1	S	0	0
8	0	0	0	1	A	0	0
1	8	0	0	0	S	0	0
1	8	0	0	0	A	0	0
0	1	8	0	0	S	0	0
0	1	8	0	0	A	0	0
0	0	1	8	0	S	0	0
0	0	1	8	1	I	0	0
0	0	1	8	9	A	0	0
9	0	0	1	8	S	0	0
9	0	0	1	9	A	0	0
9	9	0	0	1	S	0	0
9	9	0	0	1	A	0	0
1	9	9	0	0	S	0	0
1	9	9	0	0	A	0	0
0	1	9	9	0	S	0	0
0	1	9	9	4	I	0	1
0	1	9	9	4	A	1	0
4	0	1	9	9	S	1	0
4	0	1	9	0	A	1	0
0	4	0	1	9	S	1	0
0	4	0	1	2	A	0	1
2	0	4	0	1	S	0	1
2	0	4	0	2	A	0	0
2	2	0	4	0	S	0	0

Binary array of key

$\underbrace{0\,0\,0\,1}_{1}\,\underbrace{0\,1\,1\,1}_{7}\,\underbrace{0\,0\,0\,1}_{1}\,\underbrace{1\,1\,1\,1}_{15}$

$(1724)_{11} = (2204)_{10}$

I—New Input Byte

A—Add

S—Right Shift 1

A basic test is that the transformation maps a typical key set into an address set essentially as evenly as a randomly distributed key set. This would affirm the cluster-destroying effect of the algorithm. The basic test is extended by determining whether the technique results in an efficient seek factor. This factor is the number of separate access-mechanism movements required to locate a record, averaged over the total number of records in an application. It measures the combined performance of the mapping algorithm and of the overflow technique, and should be closely comparable to that of a randomly distributed set using the same overflow procedure. The overflow technique is of secondary interest here but, since one has to be assumed to develop the values for the seek factor, the "consecutive spill" routine was used. If a record is not found in its home bucket, consecutively higher-numbered locations are searched until it is found.

To establish universality, the algorithm should map key sets of widely divergent structural characteristics into a given memory configuration with essentially equal effectiveness. A given memory configuration is one in which

TABLE 3

Key Sets Used in 704 Simulation

Number	Key Set Identification	Description	Code	Symbolism*	Records in Test Set
1	A	Electrical Equipment Mfr. —part numbers	bcd	B	8192
2	B	Military Establishment —part numbers	bcd	N	8192
3	C	Business Machines Mfr. —part numbers	bcd	N	8192
4	D1	Electrical Controls Mfr. —part numbers	bcd	B	8192
5	D2	Electrical Controls Mfr. —part numbers	2/5	B	8192
6	E	State Motor Vehicle Dept. —drivers' names	bcd	A	8192
7	F1	Farm Equipment Mfr. —part numbers	bcd	B	8192
8	F2	Farm Equipment Mfr. —part numbers	2/5	B	8192
9	G	High School —student names	bcd	A	4096
10	H1	Random Alphanumeric Patterns	bcd	B	8192
11	H2	Random Alphanumeric Patterns	2/5	B	8192
12	I	Random Numeric Patterns	Binary	N	8192

*N—numeric A—alphabetic B—alphanumeric

the relevant parameters are pegged at certain values. These parameters are average bucket occupancy, $\bar{b}$ (i.e., N/K), and load factor, L. "Equal effectiveness" means that essentially the same percentage, R, of the entire key set gets housed in home buckets. For example, for $\bar{b} = 64$, and $L = 95\%$, a large data file using this algorithm can expect to have, say, 97% of its members housed in home buckets. This yardstick is meaningful and directly useful to the application planner.

To affirm code independence, the addressing algorithm must map essentially the same percentage of a key set into home buckets when the key set is expressed in different binary-base codes. On Table 3, three of the sets, D, F, and H, are expressed in both binary-coded-decimal and 2-out-of-5 notation to test this independence.

A 17-16 radix transformation was performed on all twelve sets. For each key set, the value of K was assigned seven different values:

$K = 2^k$, where $k = 1, 2 \ldots 7$; i.e., K-values of 128, 256, 512, 1024, 2048, 4096, and 8192.

A frequency distribution was tabulated relating f to b for each K. From this distribution, R was determined for a load factor of 100%, by choosing $C = \bar{b}$. Other values of R were computed for lower load factors. The latter were obtained by successively increasing the value of C, since

$$L = \frac{\bar{b}}{C}.$$

The results from the IBM 704 computer simulation were plotted in Figure 4 as a curve with R against L. A family of curves was thus generated with the parameter $\bar{b}$ taking on different values. The results for R for a given $\bar{b}$ and L were so nearly identical (root mean square deviation less than 1%) for all nine data key sets that only one curve representing their average was plotted for each value of $\bar{b}$. The R values for $\bar{b}$ of 4, 2, and 1 were similarly uniform for all nine sets but are not shown in the graph because they offer only two or fewer points in our range of interest of the load factor; i.e., 80% to 100%.

The results for the three originally random key sets were also averaged and plotted as a separate group in Figure 4. These R values also show an *rms* deviation of less than 1% related to its own group and to the entire twelve sets.

The results for three random key sets whose address mapping was performed by k-bit truncation rather than by our algorithm are also plotted in Figure 4. Again the *rms* deviation, among themselves and related to the twelve-key-set average, is less than 1%.

A set of theoretical values for R were computed from Poisson Distribution Tables,[2] using the same values for $\bar{b}$ and L as in the empirical runs. The relationship which yields R is simply:

$$R = \frac{\sum_{b=0}^{C} p_b \cdot b + C \sum_{b=C+1}^{\infty} p_b}{\bar{b}}$$

where p_b is the Poisson probability for an occupancy level of b records per bucket, and C is the capacity of a bucket adjusted to get a specific load factor $L = \bar{b}/C$.

Since all curves were quite close together, it was decided for clarity to graph only the curves for the empirical results without depicting their points. For comparison purposes, the theoretical Poisson points were shown but without their curves.

The average seek factors, S, were calculated for the nine data key sets; their arithmetic means for a given pair of parameters (bucket capacity and load factor) are plotted in Figure 5. The same operations were performed on the three random key sets.

The seek factor was arrived at thus:

$$s = \frac{K + \sum g_i}{K}$$

CONCLUSIONS

The empirical results and the hardware implementation indicate that the functional objectives sought have indeed been met.

The empirical results show that for a given set of parameters $\bar{b}$ and L, the percent R of the data set lodged into home buckets tends towards a central value with narrow deviation. These

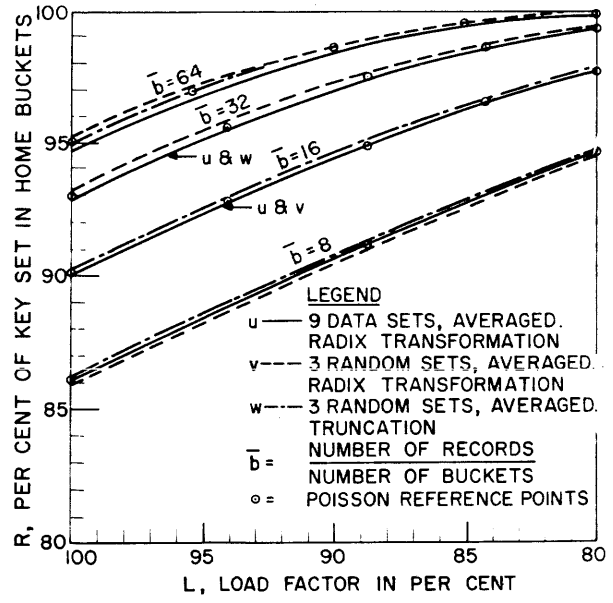


Figure 4. Comparative Mapping Efficiency of Radix Transformation Algorithm.

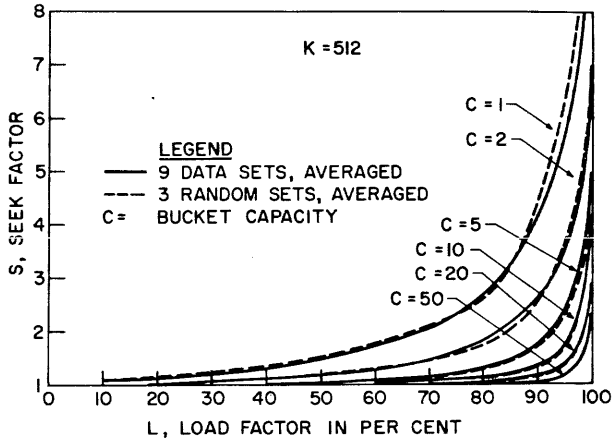


Figure 5. Comparative Seek Factor Efficiency of Radix Transformation Algorithm (using consecutive spill overflow technique).

central values were shown to be quite close to the theoretical Poisson values. This suggests then that a comprehensive table of theoretical values of R as related to $\overline{b}$ and L may be set up for the use of the application planner. Using this table to make reasonable predictions for the mapping performance of the radix transformation algorithm, he could attempt systematic trade-offs between the parameters and the performance until he is satisfied with the combination.

The simulation effort with widely differing key sets shows this algorithm to be a directly usable method that is independent of key length, source language and machine code.

Key addressing by radix transformation meets its optimal fulfillment when implemented as independent hardware with a binary address output. Augmented to provide the added option of decimal address output, it is slightly less simple but offers wider flexibility in system applications.

ACKNOWLEDGEMENTS

The author is indebted to the following individuals for auxiliary participation in the work covered by this paper: Mr. R. F. Arnold for major contributions in the early evolution of the transformation algorithm; Mr. F. Magness for assistance in the detailed logical design; and Mr. R. Togosaki and Mr. J. Barnes for preparing the several 704 programs for computer simulation and verification with customer key sets. The latter were made available through the courtesy of several IBM Sales and Service Bureau offices.

REFERENCES

1. FELLER, W., "An Introduction to Probability Theory and Its Applications, Vol. 1." John Wiley & Sons, Inc., New York, 1950.
2. MOLINA, E. C., "Poisson Exponential Binomial Limit." Van Nostrand Co., Inc., New York, 1942.
3. VINOGRADOV, I. M., "An Introduction to the Theory of Numbers." Pergamon Press, London and New York, 1955.
4. HAMMING, R. W., "Error Detecting and Error Correcting Codes." Bell System Technical Journal, 29, 147-160, April 1950.
5. PETERSEN, W. W., "Addressing for Random Access Storage." IBM Journal of Research and Development, Vol. 1, No. 2, April 1957.
6. GRIFFIN, H., "Elementary Theory of Numbers." McGraw-Hill Book Co., New York, 1954.

ADAM — A PROBLEM-ORIENTED SYMBOL PROCESSOR

A. P. Mullery and R. F. Schauer
Thomas J. Watson Research Center
International Business Machines Corporation
Yorktown Heights, New York

R. Rice
International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

Digital computers have evolved in their own technical environment, and to a large degree independently of the problem environment. Thus it was necessary to have computing centers with staffs of programmers as intermediaries between machines and users. As the inadequacy of the arrangement became apparent, problem-oriented languages were written, with compiler programs to allow the machines themselves to do the conversion to their own (machine) language. Accommodating to the nature of the computer in this way still was not the answer from the scientist's or experimenter's point of view, for there remained an enormous commitment of processing (compiling) and debugging prior to the first feedback of results. Furthermore, it proved necessary to write compilers for many problem fields, which gave this mode of solution a patchwork look. For these reasons, we decided to attack the problem at its roots by changing the fundamental nature, i.e., the organization of the computer itself. We took it as the aim of our work to allow the experimenter to use the computer as directly as possible as an experimental tool.

We reasoned that "data" upon which machines operated have certain similar characteristics even though problems in which the data are used vary. We therefore began by making a study of the nature of data. In general, data is not just of the form acceptable by most present day computers—a contiguous string of fixed length, fixed point data, but is variable field length, variable format, structured, and not necessarily contiguous. For example, a three-by-three matrix is a string of nine data symbols. But more than just a string of symbols, a hierarchy or grouping is usually imposed on the string. In this matrix example, the grouping consists of symbols which are contained in rows in a matrix. This is a simple grouping. The English language, considered as data, is grouped as words in phrases, in sentences, in paragraphs, in chapters, in books, and in libraries. No matter what one calls these groupings, the structure they indicate does exist in data. Clearly each symbol in a string of data could contain any number of characters.

Further, a string of data need not be contiguous. In text, for example, a footnote is part of a string, linked (by a symbol) but physically removed. It is necessary to have a way of storing non-contiguous data in the computer and of providing linkages. Where such data applies to many strings, it should be stored once

The research reported in this paper was sponsored in part by the Air Force Cambridge Research Laboratories, Office of Aerospace Research, under Contract AF 19(628)-1621.

and linked as necessary. An analogy is a reference in text cited frequently though given only once. Insert procedures involving the movement of entire strings of data are inefficient and should be unnecessary.

If the data to be processed has variable field length, variable format, structure, and is not necessarily contiguous, any language or notation to describe the data or operations on it must not restrict but take advantage of this form. This leads to the following implications concerning the language and the system:

1. That it be able to handle variable field length data and instructions.
2. That it be able to maintain the structure of the data internally.
3. That it be able to use and operate on this structure.
4. That it be able to handle symbolic addressing of the data.
5. That it be able to interpret links in the data correctly.
6. That it be able to insert links in the data when necessary.

In addition, in order to be as general as possible, the machine system should not impose limits on the number of levels of links, the number of names in the data, the size of the data or instruction fields, etc.

The language must include operators to permit the three basic data operations—creation, movement, and destruction. The methods by which data can be created are limitless. Some, however, are more commonly used than others. These include: Add, Subtract, Multiply, Divide, AND, OR, NOT, and Reproduce. Basic data movement operations include Insert, Separate, and Join. Data destruction is accomplished with a Delete operation. All of these operators must be defined for sets or strings of data as well as single fields. Probably the most important feature of such a high-level language is the ability to easily define new operators. This ability to define operators must be simple and flexible, and the execution of the resulting subroutines must be fast and efficient. There should be no limit on the number of levels or subroutines; that is, subroutines must be able to contain other subroutines which can contain other subroutines, etc.

The use of the language should reduce the need of a program controlled housekeeping to a minimum. As examples, the assignment of storage space should be completely automatic, and data operations should be independent of data format; that is, if the data is not in the proper format for any given operation, machine control will do the necessary conversion from any permissible format to the desired one.

In order to implement this language efficiently, a completely new machine organization has been evolved. This organization has been determined only by the above goals. This has meant new concepts in the organization of the storage, of the process unit, of the input-output, of instruction and data flow paths, of controls, etc.

The following sections describe the characteristics of the data, of the language, and of the resultant machine organization.

DATA

It was indicated in the introduction that data has structure. It may be a simple structure such as rows and symbols for a matrix or it may be the complex structure such as that in the English language. In any case the symbol is the lowest meaningful grouping—e.g., the character "b" has no meaning, but the characters "ball" do represent something and, therefore, make up a symbol. The four characters b-a-l-l are a symbol for the physical object, a ball.

No matter what these groupings may be called, they do exist within the data and are indicated by identifiers. These identifiers are usually special symbols which indicate the end of a group and the start of a new group. For example, a record mark indicates a boundary between two records. Many different such marks have been used, depending upon the names which have been given to the grouping. In order to avoid unnecessary confusion, the groups

Character	Paragraph
Symbol	Chapter
Phrase	Book
Sentence	Library

shall be used throughout. The identifiers associated with each group shall be as follows:

Group	*Identifier*	
	Use	Mention
Character	(Implied)	⑦
Symbol	①	①
Phrase	②	②
Sentence	③	③
Paragraph	④	④
Chapter	⑤	⑤
Book	⑥	⑥
Library	⑦	⑦

When the structure of a string of data or instructions is to be identified, then the "USE" identifiers are used. If some group of data is being referred to in the instructions, then the "MENTION" identifiers are used. The ① identifier will indicate the end of a symbol and the start of a new symbol. These identifiers form a hierarchy in that a ② also implies a ①, a ③ implies a ②, and a ①, etc. If more than one identifier ever appear together with no other characters separating them, all but the highest identifier will be neglected and dropped. Each symbol many contain any number of any characters. Each phrase may contain any number of symbols, etc.

The names of data will appear with the data itself. A name can be given to any string of data at any level from Symbol to Library. This string may also contain named groups of data. Thus, named groups within named groups are allowed. The name character *n* will surround the name when it appears with the data. The first character following the second *n* will be a "MENTION" identifier which will indicate the level of the data being named. If this mention identifier does not appear with the name, then the name is assumed to name data at a level indicated by the first following "USE" identifier. For example, a data string will appear in the data storage as

n ADAM *n* ④ ABEL *n* ③ EVE $\overset{s}{p}$ I *n* ② ④

----*n* EVE $\overset{s}{p}$ II *n* ②----③----②----④

Here ADAM will name the complete paragraph; ABEL will name the first sentence in that paragraph; EVE $\overset{s}{p}$ I will name the first phrase in the first sentence; EVE $\overset{s}{p}$ II will name the second phrase in the first sentence. The names themselves may be formed from any combination of any number of the general characters A - Z, a - z, 0 - 9, $_{0\ -\ 9}$, etc.

Certain operations require numeric data. Numeric data may contain any number of the digits 0 through 9 and may contain at most one of each of the characters +, −, . (decimal point), and *exponent*. Together these represent a unique number. However, such a number can be expressed in many ways. Examples of permissible data format external to the machine are as follows:

① 19 ①

① −19.76 ①

① *ex* + 02 + .1976 ①

In such numeric data, the decimal point is assumed to be at the right unless it is written elsewhere. A decimal point can occur any place in a number if the number does not also have an exponent. If the number is written with an exponent, then it must be expressed so that the mantissa is greater than or equals 0.1 and less than 1.0—the decimal point must be at the left. The number is assumed to be positive unless otherwise indicated. If, however, the number is written in exponential form with the exponent preceding the mantissa, the sign of the number must be written.

Some operations require binary data. Such data may contain any number of the characters 0 and 1. Each symbol of binary data should begin with the base 2 indicator *L*.

A data string may be linked to another data string by means of a ⑧ link character. Whenever a ⑧ is met in a string of data, the data indicated by the name following the ⑧ will, in effect, be considered to be part of the original string. A given string of data can be so linked to many data strings. For example, in the data strings

n A *n* ④ Brown ① is ① a ⑧ Des ① school ① with ① many ⑧ Des ① students ④

n Des *n* ② very ① good ②

the fourth and ninth symbols of A are both "very." Thus, the data string called A is, in effect,

n A *n* ④ Brown ① is ① a ② very ① good ② school ① with ① many ② very ① good ② students ④

PROPERTIES OF THE LANGUAGE

The language will consist of verbs, nouns, and modifiers, and rules for their use. These rules are the syntax of the language. Much of the power of the language depends on the syntax. This syntax should be simple and direct but should allow a great flexibility of use. These rules for use should be generally applicable.

An English-like structure is used. The operators are classified as nouns, verbs, adverbs, and adjectives. A noun with any number of adjectives modifying it will make up a noun phrase. The noun phrase must always indicate data contained within the extent of the name used. A verb and any number of adverbs modifying it will make up a verb phrase. An operation is a combination of at least one verb phrase and one noun phrase which accomplishes some process (for example, A + B is an operation). A sentence will consist of at least one operation. All operations in the sentence are dependent on the result of other operations in the sentence and are syntactically independent of operations in other sentences. In general, the process to be performed and where to place the result, if any, must be specified in a sentence.

Some verbs require only an object. Other verbs require both a subject and an object. Still others require any number of objects. A subject or object can be a noun phrase, an operation, or a group of operations. The subject of a verb is assumed to be the result of all operations which preceded the verb in the sentence. If a parenthetical phrase precedes the verb, then only this phrase is the subject of the verb. Exceptions to this are verbs such as *, /, and *AND*. Here normal rules of precedence apply. Any number of parenthetical phrases may be used, both with algebraic and non-algebraic operations.

The object of a verb phrase is the following noun phrase only. Again, if a parenthetical phrase follows the verb, then this parenthetical phrase is the object. If several objects are required, each object should be separated by a comma (separator). Each of these objects may be a noun phrase, an operation, or a group of operations.

All adjectives will follow the modified noun. A given noun, however, may have any number of adjectives modifying it. An adjective is considered to modify not just the noun, but the noun as modified by any adjective preceding the particular adjective. An adverb will precede the modified verb. Again, a verb can have any number of adverbs modifying it.

A name may be given to a sentence or group of sentences at any level. A name may not be given to a part of a sentence in instructions. Named sentences or groups of sentences may be contained within other, larger, named groups of sentences. A name of an instruction or group of instructions must be defined as a verb. For example, an instruction might be written as follows:

$$v \text{ Start } v \text{ ③ A + B} \rightarrow \text{C ③}$$

A named verb contained within another verb is considered to extend to the end of the highest named verb in which it is contained.

NAMES AND SYMBOLIC ADDRESSING

Names of data and instructions may contain any combination of any number of the general characters. In the machine language all data and instructions are referenced by their names. A character combination in the instructions can be operated upon only if it is interpreted as a literal. An absolute address, direct or indirect, will never appear in a program. The machine itself will assign locations in storage to named sequences of data and instructions. In order to accomplish this, a fixed length table with two characters (16 bits) per entry is provided. The address of a location in this table is derived from a name. This mapping may be simple. For example, if 4096 locations are provided, then the middle six bits of each of the first two characters of a name may be used to obtain a location in the table. Of course, any other mapping technique may be used; a hash address scheme using every character of a name, for example, may be more efficient. All names that are being used and which map to the same location in the table will be contained in a closed, linked loop in the main memory. The address in a location in the map table will be that of the name last used in the corresponding loop. Fig. 1 demonstrates the construction of such a loop. In this example, the first two characters are used to derive a location in the table. Thus, all names starting with a particular pair of characters will be contained within a loop. The "MA" loop containing the names

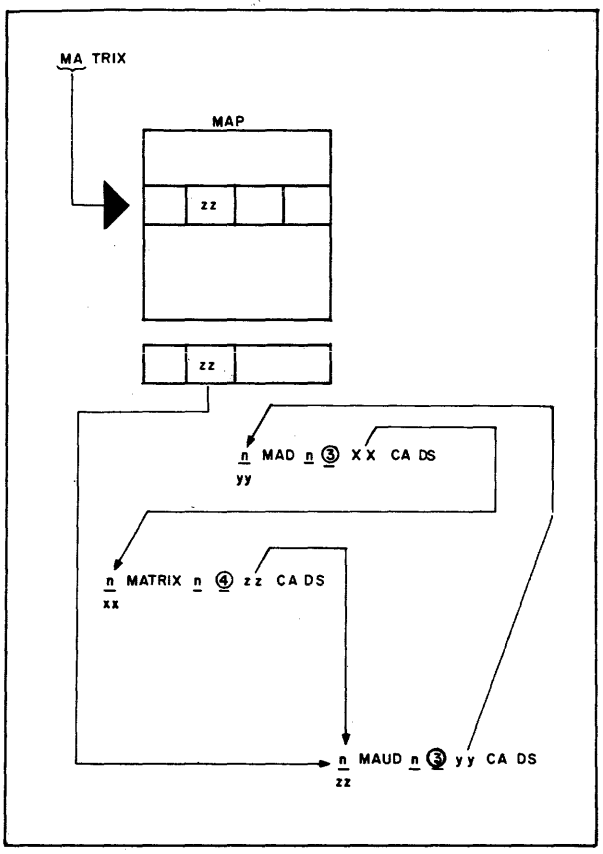


Figure 1. Symbolic Addressing Map and Name Loop.

MAD, MATRIX, and MAUD is shown. The name MAD is stored at location yy. With the name MAD, is a link address to the next name in the loop, MATRIX. Two addresses follow the name MAD to indicate the location of the "current" item and beginning of the data specified by MAD. Similarly, at location xx, the name MATRIX is followed by the link zz which is the location of the next item in the loop, MAUD. Following MAUD is a link yy, which closes the loop. In the table location is the address zz, of MAUD, which happened to be the last name used of this loop. In the example, the noun MATRIX is to be found. This name maps to the "MA" location in the table which gives the address xx. At zz, the name MAUD and MATRIX are compared. These are not equal. The name at yy is next compared with MATRIX. Since the comparison again fails, xx is accessed and the names compared. The successful comparison locates data whose name is MATRIX. If a name which is not contained in the loop, MATE, for example, is defined, every name in the loop will be compared with MATE. After going completely through the loop with no successful name comparison, the new name is placed in some available location, say ww, and a new string created. The link address associated with the first item in the loop—here zz with MATRIX—is placed with MATE and the address ww stored with MATRIX. The resulting name loop is shown in Fig. 2. Similarly, when a name is being deleted, the link addresses of the preceding and following names are adjusted to again form a closed loop. The novel feature of this symbolic addressing system is that the size and number of names is limited only by the capacity of main storage. There may be any number of names of any size in a loop. The number of loops is limited by the size of the table and determined by the names in use.

SPECIFICATION OF DATA WITHIN STRUCTURED STRINGS

Particular items within a named data sequence are indicated by means of the ⓘ ↑ j, ⓘ $ j, and ⓘ ¢ j adjectives. In the data string

n A *n* ③ - - - ① - - - ② - - - ① - - - ③

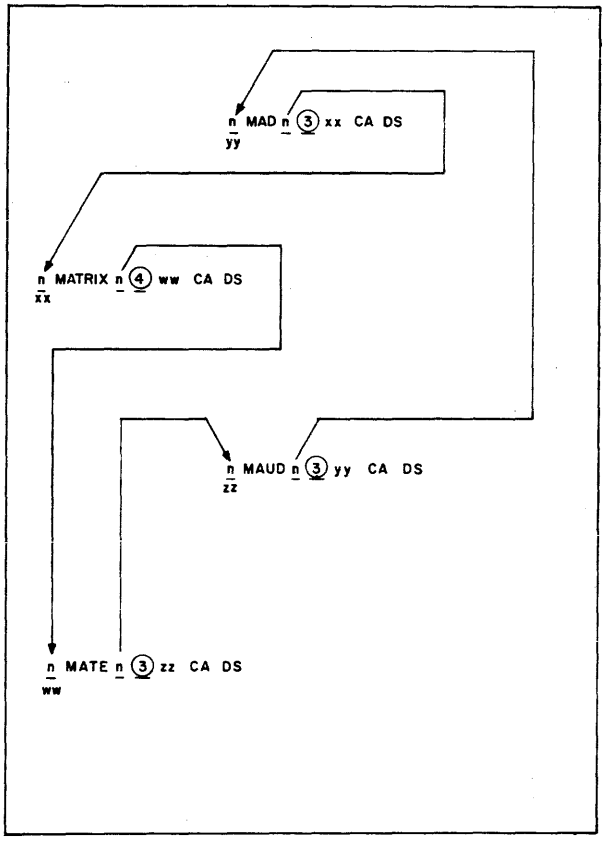


Figure 2. Modified Name Loop.

A names the whole sentence. The first phrase in this sentence is indicated by the noun phrase A ② ↑ 0. The second symbol of the second phrase is indicated by A ② ↑ 1 ① ↑ 1. In a similar manner, every part of a named string of data can be indicated. Thus, an adjective ⓘ ↑ j will indicate the jth item at level ⓘ from the point previously specified. The ⓘ ↟ j adjective has the same effect except that it will also set a "current" indicator at the item referenced. Thus, the noun phrase A ② ↑ 1 ① ↟ 0 will specify the first symbol of the second phrase of A and mark this symbol current item of the data sequence named A. The adjective ⓘ ↞ j will find the jth item at the ⓘ th level following the "current" item and set this new item as the "current" item. If, for example, succeeding symbols in a string are required, these may be indicated by the noun phrase A ① ↞ 1. Each time this noun is interpreted, the next item in A will be indicated without any indexing or other change to the program. In order to perform this type of addressing, the general form of memory organization shown in Fig. 3 will be used.

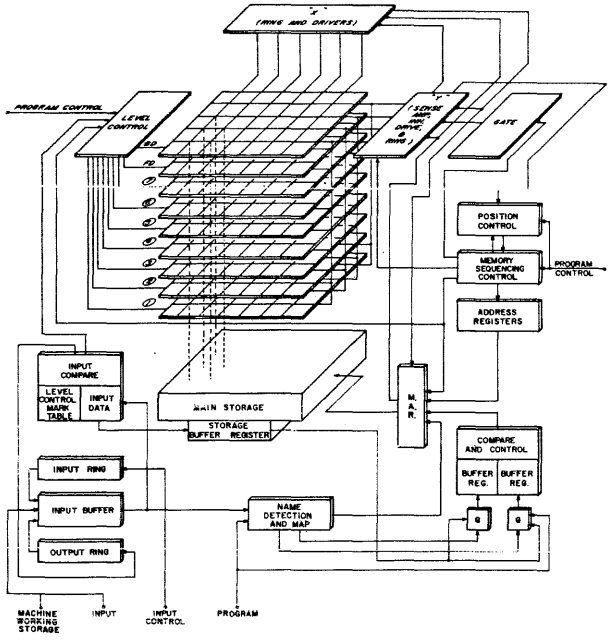


Figure 3. Special Memory Organization.

As data or program is input into the machine, it will pass through the input control. The input control will break up the information into machine words for storage in the machine. The beginning of each new classification of data, or every data identifier, will always begin a new machine word. The input control will scan the input for data identifiers and form the variable length block into machine words for storage. The input control will also have partial control over the special control planes associated with the memory. There will be one control plane associated with each data identifier to be used in the system. Whenever an identifier is detected in the input information, a core will be set in the plane identified with this character in a position equivalent to the address in which the machine word containing this character is being stored. Similarly, a core will be set in the same position in all lower levels of identifiers explicitly stating what is implied in the data. If an identifier is detected before the storage buffer register is filled, *null* characters will be inserted to fill out the machine word, and then the word is stored. The identifier will be held by the input control until the storage buffer register is again ready to accept data and will be inserted as the first character of the machine word being formed. When this word is ready for storage, the appropriate related cores will be set in the special control planes. All information will be input in this manner.

The name symbol, verb symbol, and key symbol will also start new machine words when detected by the input control, and marks are set into a special core plane. The use of this mark will be discussed later.

Normally data will be stored in 8-bits and parity outside of the machine whenever possible. This same representation will carry over to internal storage for all alpha-numeric symbols. Certain symbols—those marked as numeric or logic—will be packed for more efficient operation. The input control must detect a numeric field as defined in a previous section and set up the normalization operation. Packing will occur during this latter operation. Logic data will be indicated by a special character. An operation similar to the normalization procedure will be initiated by the detection of this special character by the input control.

With this structure of data stored in the memory, it is possible to find any particular item in a string relatively quickly. Let us say we are looking for

$$A\ \underline{④}\uparrow 2\ \underline{③}\uparrow 1\ \underline{②}\uparrow 2.$$

The machine will first find the beginning of the string named A in the manner described previously. Before attempting to describe the search for the specified part of A, consider the organization of the special core planes. There will be one core plane for each data identifier indicating one level of classification. Each core in this plane will represent one machine word in the main store. The special core planes will be used in a 2-D memory organization so that it will be possible to gate out the contents of 128 cores in one read-write cycle. If a core has been set to a "1," this implies that that level identifier or one higher occurs in the associated machine word. Therefore, by using appropriate circuitry, one is able to scan the contents of a row of one of the special planes from either right or left to determine which of the 128 machine words contain the identifier in question.

The above discussion has assumed that the extra core planes were a separate entity. It is also possible to select a portion of the main store for assembling this information. Even though the main store has a 3-D organization, it is possible to organize it such that the location of identifiers in blocks of machine words (probably 64 at a time) is available in one read-write cycle. This is the organization proposed for the system being designed.

The search for the location of the data of the above example would proceed as follows:

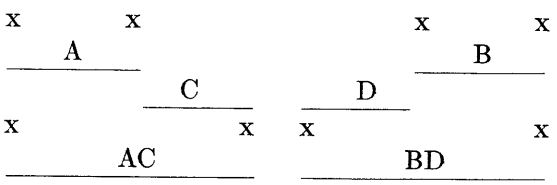
1. Locate the beginning address of the block called "A."
2. In the ④ level plane read out the row in which a location corresponds to this address.
3. Begin the search for ④ marks at this address. As a ④ mark is encountered, count this mark and compare the total with that obtained from the adjective evaluation. Continue the search if the two numbers do not agree.
4. If three ④ marks (magnitude of number in adjective incremented by 1) have not been located and counted by the end of the row, increment the appropriate rings and read out the next row.
5. When three ④ marks have been counted, store the address associated with the last one located. Then read out the ③ level row containing this address.
6. Begin searching in the above manner for the second ③ mark starting at the address of the third ④ mark. When this is located, store its address and proceed at the ② level.
7. When the address is finally located, the machine will look to the instruction control to determine what to do next. It is possible to operate on any information which can be located in the described manner from this newly located point in the data.

The absolute address of the current item in a data string is stored with the name of the string. Thus, when the adjective is ① c j, the scan is begun at this address; and when the described point is located, the current address will be changed.

In order to perform the input procedure described in this section and also to perform some operations such as insert and delete which will be described, the memory organization must be capable of determining and remembering blank memory locations. A special core plane is included to perform this function. The extent of a blank sequence is determined before the location of the sequence is placed in the blank plane.

The special plane will be accessed in exactly the same manner as the other special core planes. In this case a core associated with the address of the beginning of the sequence will be set and one associated with the address immediately following the last address of the sequence will also be set. This technique permits one to combine adjacent blank sequences. This will be accomplished by setting an addressed core in the special plane to a "1" if it previously was a "0" and vice versa.

Consider the following example:



Two blank sequences exist in memory and are labeled A and B. Two additional sequences, C and D, are to be added. When the initial address of C is to be set, the control will set the equivalent core to a "0" because it was previously a "1," indicating the end of A. When the end of C is set, the sequence now contains both A and C. In a like manner, when D is put into the special plane, the end of D removes the beginning of B and gives a single sequence.

A blank sequence will be available to memory control at all times. Information relating to the address of the next blank word and the address of the last word of the sequence is kept by the memory control. When a blank sequence has been filled, the memory control will initiate a search of the special plane starting at the address of the end of the last sequence. The address of the next mark in the special plane will be the address of the beginning of the next blank sequence. The second mark will be the end of the sequence. These addresses will be stored in the memory control. Thus the memory will be loaded in cyclic manner in order to reduce the length of time necessary to locate the next blank sequence.

SEARCH AND STORAGE OF VARIABLE LENGTH DATA

All data has been considered to be of variable length. In addition, the programmer, through such instructions as *insert, join, delete, define* can change the length of a particular string. It is not desirable, when the length of the string is changed, to move any portion of the string in order to make it continuous. Therefore, a means of linking disjoint parts of a string will be provided.

This link will be composed of three characters, a special "go to" character indicating a link and a two character address locating the next word in the sequence. This group of characters will be placed at the end of the machine word which would normally precede the linked data. If this machine word contains information other than nulls in any of these three character positions, the characters are extracted and stored in an available word. The data to be linked is either stored in the following blank locations or another link is provided between the "prelink" word and the linked string. A link is placed at the end of this linked data back to the original string.

A special link symbol, ⑧, permits the programmer to insert common data, which is stored once, into several strings. When this character is recognized by input control, this special character will be interpreted as an identifier in the sense that it forces a new machine word to be started with this character. In this case the character is stored in the first and sixth character positions of the new word, and the nulls are stored in the rest of the word. The name following the ⑧ character will be stored beginning in the next word. The first time that this data is used and the symbol interpreted as a link, the name identifying the common insert will be used to determine the address of the beginning and end of the data. These addresses will then be stored with the ⑧ link. The name will not have to be referred to in subsequent uses of the original data. At the end of the inserted string, will be placed a variable link. When this string is being used, the address of the next location in the calling sequence is placed in this variable link. Consequently, the link back to the calling sequence is provided at the time of reference.

Since a link indicates a discontinuity in a string of data, the location of a discontinuity must be available when the contents of an identifier plane are being searched and counted as described previously. Another special core plane —the Forward Discontinuity Plane—is provided. This will be accessed and searched in parallel with any of the identifier planes when a forward search is underway. A bit in this plane set to "1" indicates that a special circumstance applies to the corresponding machine word, and the machine word must be accessed before any search can continue. In general, a discontinuity will be indicated in that machine word and the search will be continued at the place indicated by the link address.

It is also possible to search backward from any point within a named string of data. However, only forward links are provided in the data. A push down store called the Reverse Push Down Store (RPDS) is used to store the reverse links generated as a forward search is conducted. Since one cannot access any piece of data beyond the extent of the name indicated, it is normally true that a forward search must

precede any backward search and thus the reverse links are always provided in the RPDS. A Reverse Discontinuity Plane will also be provided to indicate the special circumstances that are pertinent to a backward search.

Let us say we are given the string A stored in memory as shown in Fig. 4. In this string there is a discontinuity at location (yy − 1). This is indicated in the forward and reverse discontinuity planes. The beginnings of the strings are indicated in the reverse plane. The ends of the strings are indicated in the forward plane. It is required to find

$$A\ \underline{②}\uparrow 3\ \underline{①}\uparrow -2$$

Once A has been found, the ② and the forward discontinuity plane are accessed and the count begun. At yy − 1 a discontinuity mark is reached. The machine word is accessed for the link address xx and the address (yy − 1) placed in the RPDS. The ② and forward discontinuity plane are accessed at xx and the search continued. At the end of the linked string, the same situation is indicated. The address zz is placed in the RPDS and the search is resumed at location yy. The second ② mark beyond but including this point is the one indicated by A $\underline{②}\uparrow 3$ (or determined by a count of the ② marks as we have been searching). The search is then continued at the one level and in a reverse direction as indicated by the next adjective $\underline{①}\uparrow -2$. At this time, the address zz is at the top of the RPDS and the address (yy − 1) below it. The ① and Reverse Discontinuity Plane are accessed and the search and count are continued backwards. Again at yy − 1, a discontinuity is searched, but this time the link address is obtained from the RPDS. Thus, the search is continued at zz. The next ① is the one desired so the search stops at this point and the RPDS is cleared since there are no further adjectives modifying the noun A.

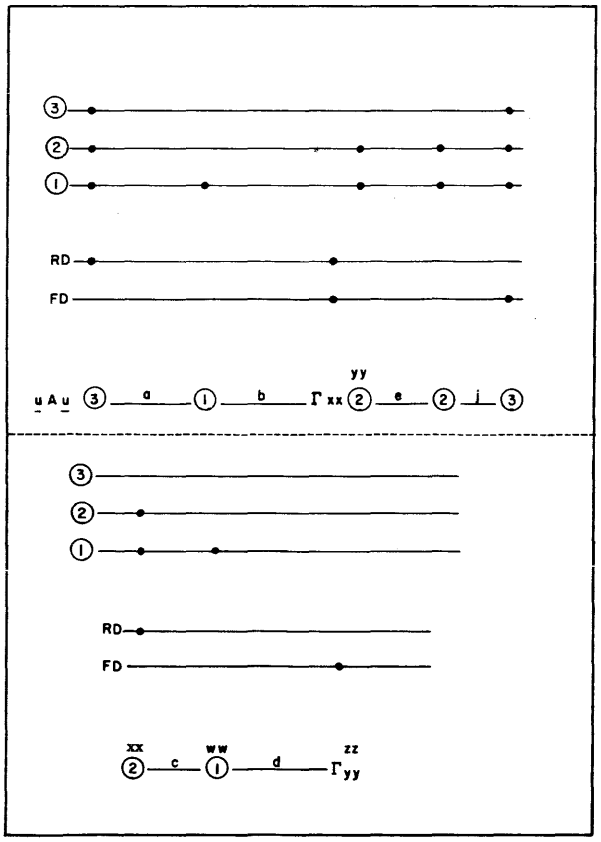


Figure 4. String of Data Stored in Memory.

If the noun phrase had been,

$$A\ \underline{①}\ c\ 0\ \underline{①}\uparrow -2$$

and the current address was ww, a slightly different situation exists. After the name is located, the first adjective indicates the current symbol. This address is determined. The next adjective specifies a backward search from this point. The ① and Reverse Discontinuity planes are accessed and the search initiated. However, the search will stop at xx, and the word accessed. In this case, the controls will determine that this word is the beginning of a linked string of data but that the RPDS is empty. Consequently, a forward search out to this point must be accomplished so as to build up the RPDS. The backward search can continue as soon as this operation is complete.

ARITHMETIC OPERATIONS

One of the considerations used in the specification of this system was that all data must be considered to be variable field length. In this situation, numeric processing becomes more difficult when considered in the conventional manner.

A number of numeric operations are basically high to low order operations. These include input, output, division, comparison, and normalization. Considering the problems of implementing these operations together with those of the arithmetic operations, it was decided to process all data high order to low order. The effect of the extra complexity that

may result in the process unit is hopefully offset by the increased efficiency of the system.

Basically, the addition of two numbers serial by character is quite similar when done either high order to low order or vice versa. In the latter case, two digits are added producing a sum and a carry or no-carry. The carry, no-carry information is stored for use in the next cycle—the addition of the next higher order pair of digits. In high to low order operation, the results of an addition are still a sum digit and a carry, no-carry indication. In this case, however, the sum digit is stored until the next cycle so that it can be modified by the carry, no-carry result of that operation.

However, there is a complication in high to low order processing. It can best be illustrated by an example. Consider the addition of the following pair of numbers:

$$\begin{array}{r} 24444\ n \\ +35555\ m \\ \hline \end{array}$$

where n and m are any digits. Proceeding from left to right, the first sum is 5 with no carry. The 5 is placed in a buffer and the next pair of digits added. The result is a 9 with no carry. The 5 is now gated to a temporary store and the 9 into the buffer. The following cycle also produces a 9 and no carry. The previous sum is put into the temporary store and the nine into the buffer. This procedure continues until the last cycle. If m + n produces a carry, this carry will not propagate through the string of nines formed and modify the first non-nine digit to the left of sequence in the result. Thus, the string of nines and the next higher order digit must be available to be changed by the carry as it is propagated.

One technique which accomplishes this is to avoid outputting any nine or digit which is in the next higher order position above a nine until it is known whether or not there is a carry to be propagated. This involves storing the next higher order digit (NHOD) and the ensuing consecutive nines until a non-nine digit is produced. At this point, the carry, no-carry information is consulted, and the NHOD and the nines are outputted after being incremented by one or zero.

When the first nine in a string is produced, the NHOD is already in a one digit temporary store awaiting carry, no-carry information from that cycle. To avoid outputting anything until a possible carry propagation can occur, the nine output is counted, and the NHOD output is inhibited. Since only nines need to be stored in this manner, a simple counter is sufficient to keep track of consecutive nines.

Finally when the output of the process unit is a non-nine digit, the NHOD is outputted after being modified by the carry, no-carry information of this cycle. The counter is then decremented by one and a nine outputted. This digit is also modified by the carry, no-carry information. This procedure is continued until the counter is cleared. The non-nine digit is placed in the temporary storage, and the processing continues on the next pair of digits.

Basically, then, the procedure is to count the nines as they are produced to inhibit all process unit output until a non-nine is detected, and then to correct for the carry as the digits are outputted.

High to low order subtraction offers no additional complexity over that present with addition. Subtraction may be accomplished by complementing the subtrahend and adding the result to the minuend. If the nines complement is used, the addition proceeds with exactly the same rules for nines handling and carry, no-carry modification as was used for addition. If the difference is in non-complement form, a 1 must be added to the lowest order digit to take care of the end around carry. If the 10's complement is taken, zeros must be counted in the same manner as the nines were in addition, and the output must be decremented by zero or one to effect the carry, no-carry propagation.

Complemented answers will be avoided by always subtracting the smaller magnitude number from the larger. If this condition does not exist initially, the roles of the minuend and the subtrahend can be interchanged by changing the true-complement sense of both operands and changing the sign of the result.

The need for a reversal can be established only after determining which operand is larger, which in turn depends upon the magnitudes of the highest order not equal pair of digits in the operands. The relative magnitude of the first pair of unequal digits sent to the process unit will determine if a reversal is necessary when using high to low order processing. The results obtained from higher order pairs of equal digits

(if any) will be the same whether a reversal occurs or not. Thus, if a reversal is necessary, only the digit cycle in which the need for reversal is detected must be re-run. With or without reversal, the remainder of the subtraction is completely normal. In either case, only one pass through the process unit is necessary. This is a distinct advantage of high to low order processing.

Multiplication and division offer no additional problems. Multiplication may be done either by repetitive addition or by multiplication of multiplier and multiplicand digits in an N-tupler and by addition of the result to the partial product. In either case, the addition will be performed as described above. If N-tupling is used, the multiplication itself is insensitive to the direction of processing. Similarly, division may be done by repeated subtraction, accomplished as described above, or by the estimation of a quotient digit, multiplication by the divisor digits, and subtraction of the result from the dividend. The subtractions may be performed as previously described, and the multiplication is insensitive to the direction of operation.

The numeric fields to be processed normally are in normal form for an arithmetic operation. Normal form in the system is considered to be

$$exp + - - + - - - - -$$

or exponent character, sign, two digit exponent, sign, implied radix point and mantissa. Usually numeric fields will be normalized during the input operation. A numeric field is a field containing only numeric characters (this includes exponent character, radix point and signs). However, a radix point must occur if the field is to be stored in normal form.

Addition and subtraction can be done on integers without forcing the operands to be in normal form. This ability is included in the system to make indexing as efficient as possible. However, both operands must be in this form, or both will be normalized before the operation begins. The operands must be normalized for multiplication and division.

A two accumulator system is used with the process unit. High to low order multiplication and division are such that the additional controls necessary for use with a single accumulator require enough extra hardware and slow the operation enough that a second accumulator is justified. (This second accumulator also speeds up some of the other automatic housekeeping and data handling chores in the system and could be justified nearly on this account alone.)

EXTENSION OF THE LANGUAGE

The most important property of the machine language is its ability to be expanded. This ability is accomplished through a means similar to the method used to construct subroutines in present day computers. However, once an operation is defined, it can be used in the same way any other operation in the language is used. There are no special rules for the use of these defined operations.

In the language some verbs, modifiers, and certain nouns have been defined and each assigned a special symbol. However, any combination of general characters may be used not only as a name for some data but also for some sequence of instructions. The only requirement for a particular combination of letters to be used as the name of an operation is that this combination not be used with any other meaning either as an operation or as a noun; that is, it must have a unique definition.

New operations may be defined using the previously defined operations and nouns. In addition, the noun "*op* n" is available for use in communication between the definition and the use. This noun will indicate the nouns included in the calling sequence. If any one noun in the calling sequence is required—the nth noun, for example—then this is indicated by writing *op* n. As an example, let us define some verb "triple add" as follows:

$$v \text{ triple add } v \text{ ③ } op\ 1 + op\ 2 + op\ 3 \text{ ③}$$

The instruction

$$\text{③ (triple add, A, B, C)} \rightarrow \text{D ③}$$

will place the sum of A, B, and C into location D.

The simple rule for the formation of a defined operation is that the last sentence of its definition must be able to replace the calling expression without violating any syntactic rules. In this replacement, the first and last identifiers in the last sentence of the definition are neglected. In the example above, the rule was obeyed since ③ (A + B + C)→D ③ is a complete sentence.

In defining a new operation, it is permissible to use other defined operations. An *operand* noun may, in turn, refer to another *operand* noun. For example, let us define the operation sq $\overset{s}{p}$ rt as follows:

```
v sq p rt v ④ op 1→Xi
            ③ (Xi→Xj
            ② .5 * (Xi + op 1/Xi)→it)
              until (it − Xj) abs<ϵ
            ③ Xi
            ④
```

Further, we can define the operation sq $\overset{s}{p}$ root using sq $\overset{s}{p}$ rt as follows:

```
v sq p root v ③ (sq p rt, op 1)→op 2 ③
```

Thus, the instruction ③ (sq $\overset{s}{p}$ root, A, B) ③ will obtain the square root of A and place this in B.

There is no limit to the number of operands nor to the number of levels in a defined operation.

A Subroutine Push Down Store (SPDS) will be used to store the addresses to which subroutines must return when they are completed. Whenever a name in an instruction is found to be a defined operation (surrounded by *v* marks), the address of the name location in the instruction is placed in the SPDS and control is transferred to the indicated subroutine. When the subroutine is finished, control is returned to the point indicated by the top address in the SPDS. Because there is no limit on the size of the SPDS (other than the size of main memory), there is no limit on the number of levels of subroutines. A defined operation may use a defined operation, may use a defined operation, etc. These addresses in the SPDS may also be used to obtain the appropriate parameter whenever an "operand" noun is mentioned. A special table, however, will be used for this purpose in order to make the interpretation of subroutines more efficient. Whenever a defined operation is found, the absolute addresses of its parameters are placed in a table. When a subroutine is first encountered in the original program, the first parameter is placed in location 11 (corresponding to the first parameter, first level), the second in 21, etc. If another subroutine is reached within the first subroutine, the absolute address of its parameters are placed in locations 12, 22, 32, etc. If an operand noun, say *op* m, is reached in subroutines at level n, then the absolute address corresponding to this noun can immediately be found at location m, n in the table. The 0, n location at each level is the absolute address of a temporary result string used at the particular level and addressed at each level by the noun *temp*. The size of this special table will limit the number of parameters and number of levels at which this interpretation can be efficiently performed. The table used in this system will allow 32 levels and 9 parameters at each level to be evaluated efficiently.

INSTRUCTION SEQUENCING

As discussed in previous sections, instructions in the system language are sentences within some defined operation. If the defined operation is at level 0 (the main program in the usual sense), the system will step through this operation, sentence by sentence. Each sentence in this case must be syntactically independent and complete. The execution of the sentences will continue to the end of the sequence in which it is started unless a programmed transfer to another defined or named operation is encountered. This is the usual transfer encountered in a branch instruction in a conventional system. If another defined operation is used in a noun phrase in a sentence, control will be transferred to that operation and returned to the sentence as soon as the operation has been completed and the resultant noun phrase evaluated. In this case, the detection of the completion of the execution of the operation and the return are automatically determined and need not be programmed explicitly.

The proper sequence of the operations is accomplished through the use of a push down store. This store will serve as a means of reversing the sequence of any part of an instruction when this is necessary for proper evaluation. This storage will also save the starting point of any recursive loop in the program. The effect of each operator and operand on its sequencing depends on the verb itself and its context. The store, itself, will consist of a fixed sequence of memory locations. Since these will be sequential, no linkage is necessary. However, if at any time more storage is required,

as many available machine words as necessary will be automatically added to the push down store. At the beginning of each overflow word, will be a link to the location of the previous machine word in the push down store.

This push down store will have the usual purpose of a "last in-first out" storage which, in effect, reverses the sequence of items from the order in which they are read in.

As an instruction is being interpreted, several quantities are accumulated and may need to be retained. As an example, the extent of an operand (beginning and end addresses of its location in the storage) is determined when that operand is mentioned in the instruction. This information is normally stored in address registers associated with the subject and object register storage. If the operation using this operand cannot be executed at this time, the operator and operand specifications must be placed in the push down store. Some special characters such as "(" or "③" must have their machine word and character addresses stored as well as the characters themselves. Consequently, each machine word in the push down store will contain only one entry; that is, operand, operator, or special character.

As the instruction is executed, temporary or partial results occur and must be stored. These results will be placed in a temporary store which is constructed similarly to the push down store. The words of this store are sequential and can be added to from main memory. This store is useful in that it eliminates machine map, search, and delete operations for these resultant operands. The extent of the operand is noted and retained when the items are placed in this store. The items are removed from the store and the space made available automatically as the sentence execution proceeds.

At the end of a level 0 sentence, both the instruction push down store and the temporary store are cleared.

AUTOMATIC INPUT AND OUTPUT

Since the programmer has no control over where data is stored in a proposed system and how much of the store a given block of data occupies, the machine must have control over some input and output operations. Part of this control is accomplished through a register in which is stored the current size of the availability list. This availability list is a linked list containing all the available locations in the store. The contents of the register are used to determine when additional space is needed in storage or when there is sufficient space to allow another block of data or instructions to be loaded into memory. Thus, when the main store is approaching some fraction of full capacity, it will output data. When the memory is relatively empty, it will input data. Once data has been in the machine, the link address associated with the names will always permit the machine to find it and return it to the main store if necessary. The programmer need only provide the machine with an input list indicating where named blocks are located and a priority, and a "scratch paper" list indicating where data should go temporarily and a priority. The final output of results will occur as programmed.

The input list will have a structure similar to the data itself. It will be a sentence whose symbols contain the name of the string of data to be inputted and the indication of where in external storage the data is located. Those strings of data that have the same priority are grouped into phrases. Thus, if two strings of data will be required at a given time, they are given equal priorities by placing them in the same phrase. The priority of the phrase is indicated by the order in which it occurs in the input sentence. Each symbol in the input sentence is of the form:

Alpha ③ *unit* 1

where Alpha is the name of some data string and *unit* 1 is the name of the external storage. The identifier mention is used to indicate the level of the named block to be input if the whole block is not to be moved at one time. A typical input sentence might be:

③ Alpha ③ *unit* 1
① Beta ④ *unit* 1
② Gamma ③ *unit* 2
② Matrix A ② *unit* 2
① Matrix B ② *unit* 1
② Dictionary ④ *unit* 1
③

The input list will be used when the memory load is some fraction of full capacity. The automatic input will continue until the memory reaches some fraction of capacity above which

the input will cease at the next permissible point. The particular values have been chosen to be .2 and .6, respectively.

Data can also be inputted into the machine by instruction. First, if a named string is defined as being the contents of an external store, the input of the contents must occur. Next, if the operand specified in an instruction is not in the machine but can be located by the system, enough data must be scanned and sufficient data inputted to locate the operand. During automatic input, the normal program execution continues and is interrupted only by the priorities assigned to machine operations for use of the memory. The define instruction which causes an input also includes the point beyond which the normal program cannot proceed until the input is completed. This permits simultaneous input and compute, with as little wasted machine time as possible. Compute must stop while an operand is being located.

If part of a string of data is inputted, a link will be placed at the end of the stored data to the external storage where the rest of the data is located. The exact form of the address will depend upon the organization of the data in the external store.

The input list always has a current address associated with it. This address indicates the data which is to be inputted next. When the data indicated in an input phrase has been completely inputted, the "current" indication is moved to the next phrase on the list. Thus, when an input is called for, the current phrase will contain the names and locations of one or more data strings. These strings may be:

1. Completely in the external storage and none yet inputted into the main internal store.
2. All partly inputted into the main store.
3. Some completely inputted into the main store but some only partly inputted.

The scratchpad list is used to specify temporary external storage for data and intermediate results during a computation. This list has the same characteristics as the input list with the exception that entries are placed on the list by programmed statements. This list is used in much the same way as the input list in that there is some fraction of full memory capacity above which output will occur and some fraction of full capacity below which this output will cease. Provisions are made to include the proper linking so that all parts of the data, whether in internal or external stores, are available to the system.

The final output of data must be programmed. When an external store is used as the object of a define verb, a point in the program beyond which the program execution cannot proceed is also included to permit simultaneous output and compute. If both programmed input and output are happening at any given time, the machine will stop at the first specified "wait" point until both operations have been completed.

CONCLUSION

The sections of this report have described a machine language and organization for an experimental digital data system. This system has been developed and designed in an attempt to provide the advanced or experimental programmer with a tool which can be used to solve many of his problems more easily and efficiently. It is expected that experience with this system will point out areas in which more work needs to be done and others in which the problem has been overestimated. It is anticipated that from such experience more sophisticated and perhaps simpler ways of solving the same problems that have been tackled here will be found. However, it is felt that the first step had to be taken no matter how faltering or short it may be. The actual evaluation of this step now awaits the finish of the logical design and the subsequent construction of this system.

ASSOCIATIVE TECHNIQUES WITH COMPLEMENTING FLIP-FLOPS

Edwin S. Lee
Burroughs Corporation
ElectroData Division
Pasadena, California

INTRODUCTION

An associative memory is one in which a word is found by specifying some of, or all of, its contents rather than its storage location. For instance assume that the words 000, 101 and 111 are stored in an associative memory with their storage locations unknown. Specifying 101 to the associative "address register" causes the location containing 101 to respond. Any additional information stored in that location may also be read out.

The associative memory is attractive from the logician's point of view. It can perform logical operations such as table look up and sorting in far fewer steps than are required by normal techniques.[1, 5, 14-17] In fact for large word arrays, operating steps may be reduced by several orders of magnitude.

The main roadblock to the acceptance of associative memories has been the high cost of the basic building block: the associative cell. Apparently only magnetic or cryogenic cells might be produced at attractive prices.[1-10] This hasn't been done yet and there is little indication that a large associative memory can be built in the near future with either technology. Both approaches contain engineering difficulties still to be overcome.

An unfortunate by-product of this technological problem has been an overemphasis on the engineering aspects of the associative memory. Only recently have significant suggestions been made with regards to computer applications.[13, 17] Even these suggestions are only a good beginning.

This paper deals with semiconductor associative cells. There are two reasons it does so. First, to demonstrate that there is a technological tool with which any conceivable associative memory, large or small, can be built today. Second, to show that this technology is a possible source of economically practical cells. Every characteristic of solid state associative cells from component tolerances to potential quantities lends itself to low cost integrated circuits.

This report is divided into two sections. In the first section, the associative cell is investigated in detail. Its theoretical properties are outlined and circuits which implement these properties are analyzed. The circuits dealt with are complementing flip-flops. They are by no means the only circuits or even the simplest circuits. The second section described the organization and operation of associative matrices in detail. Operating difficulties and their solution are illustrated with several examples.

THE BINARY ASSOCIATIVE CELL

The binary associative cell is a circuit which combines memory and logic. Its output signal is a function of its own information content and of the information stored in a second circuit called a compare cell.

The binary associative cell can store either a 0 or a 1. The compare cell can store a 0, a 1, or a Ø ("Don't Care"). When the compare cell stores a 0 or a 1, the output of the associative cell is true only if it stores the same information and false if it doesn't. When the compare cell stores a Ø the output of the associative cell is true regardless of its information content. The truth table of Figure 1 summarizes this logical relationship.

CONTENTS OF COMPARE CELL	CONTENTS OF ASSOCIATIVE CELL	ASSOCIATIVE OUTPUT
0	0	TRUE
0	1	FALSE
1	0	FALSE
1	1	TRUE
Ø	0	TRUE
Ø	1	TRUE

Figure 1. Binary Associative Cell's Output Truth Table.

The associative cells considered here are basically complementing flip-flops like the one shown in Figure 2. When the flip-flop of Figure 2 is in the 1 state, $Q1$ is in saturation and $Q2$ is off. When it is in the 0 state $Q2$ is in saturation and $Q1$ is off. The collector voltage of the transistor which is off is several volts negative. The collector voltage of the transistor in saturation is near ground.

The write terminal can be used to set the flip-flop to either stable state. It is normally at ground potential. A negative voltage pulse applied to the terminal forces $Q2$ on and sets the the flip-flop to the 1 state.

A positive or negative voltage pulse of proper amplitude and duration applied to the input terminal complements the flip-flop. If the cell is in the 1 state prior to an input pulse, it is in the 0 state after the pulse and vice-versa. The input terminal is normally at ground potential.

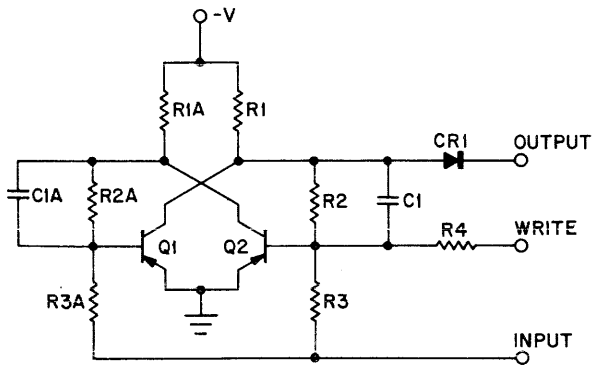


Figure 2. The Basic Cell.

The output of the cell is taken at the collector of $Q1$. It is defined as true when $Q1$ is cutoff (when the cell is in the 0 state) and false when $Q1$ is in saturation (when the cell is in the 1 state).

The logical output response described for an associative cell is produced with the complementing flip-flop in a two phase system. In phase I, a flip-flop cell is in a state determined purely by its information content. In phase II, its state is determined both by its information content and by the information to which it is being compared. That is, it is in the state which produces the appropriate associative output.

The transition from phase I to phase II involves an operation on the cell which produces the correct logical output. The transition from phase II back to phase I involves an operation which restores the information content of the cell. These operations are determined solely by the information to which the cell is compared and they are enabled through the input terminal.

Figure 3 shows the basic arrangement between the associative cell of Figure 2, the comparing circuit, and a clock. The associative cell produces the "match" or "mismatch" indication, the comparing circuit contains the information being compared to that of the associative cell and the clock produces pulses to drive the system from one phase to the other.

The clock pulse generator produces two negative going voltage pulses. These pulses are suitably separated timewise and are of proper amplitude and duration to cause the associative cell to complement when and if they reach its input.

Prior to the first clock pulse, the system is in phase I and the state of the associative cell is determined solely by its information content. If it is in the 0 state, its output is true, if in the 1 state its output is false. The compare flip-flop is likewise in a state determined by its information content. If it contains a 0 its 1 output (the one going to the "AND" gate) is false. If it contains a 1 then its 1 output is true. The compare flip-flop remains in its initial state throughout the sequence.

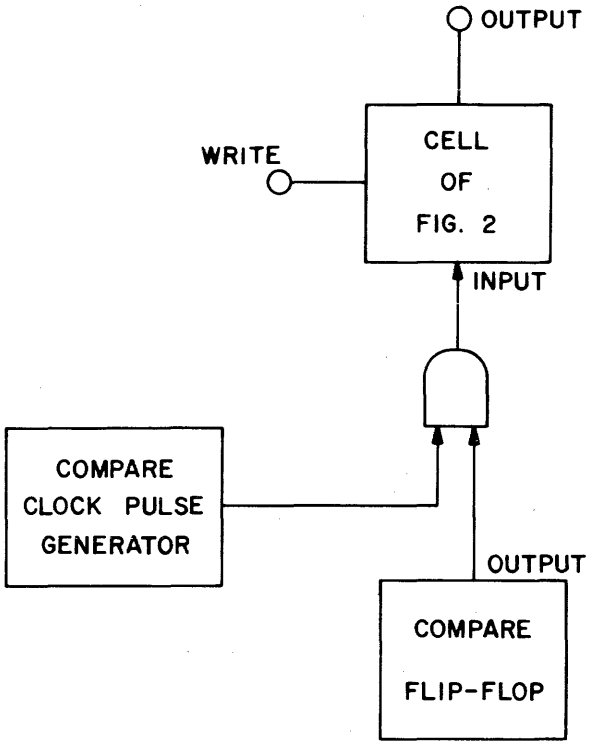


Figure 3. Associative Cell with Compare Controls.

The first clock pulse complements the associative cell only if the compare flip-flop is in the 1 state, since both inputs of the "AND" gate must be true. If the compare flip-flop is in the 0 state, the output of the "AND" gate is never true and the associative cell remains in its initial state. Figure 4A shows what the clock pulses accomplish. In phase II, after the first clock pulse, an associative cell is in the 0 state if in phase I it contained the same information as the compare flip-flop.

After the phase II output has been sensed the second clock pulse occurs. Since the state of the compare flip-flop has not changed, the associative cell complements again if it complemented before, and does not complement if it didn't. Since an associative cell has either complemented twice or not at all during the two clock pulses, it is in its phase I state after the second pulse.

	INITIAL STATE OF COMPARE FLIP-FLOP	INITIAL STATE OF ASSOCIATIVE CELL	ASSOCIATIVE CELL OUTPUT PHASE I	PHASE II	PHASE I	FINAL STATE OF ASSOCIATIVE CELL
[A]						
COMPARE CYCLE	0	0	T	T	T	0
	0	1	F	F	F	1
	1	0	T	F	T	0
	1	1	F	T	F	1
[B]						
WRITE CYCLE	0	0	T	T	T	0
	0	1	F	T	T	0
	1	0	T	T	F	1
	1	1	F	T	F	1

Figure 4. Logical Output Signals of the Associative Cell in Phase I and Phase II for Compare and Write Cycles.

At this point one method for writing into an associative cell will be described. In this writing mode, the information contained in a compare flip-flop is reproduced in a selected associative cell. The cell is selected by pulsing its write line during phase II.

The information to be transferred is stored in the compare flip-flop. The first clock pulse is generated. After this clock pulse the write line of the selected associative cell is pulsed so as to set the cell to 0. The second clock pulse occurs. After the second clock, the cell contains the same information as the compare flip-flop. This sequence is shown in Figure 4B along with the initial and final states of the associative cell.

As far as the compare flip-flop and the clock are concerned, there is no difference between the compare cycle of Figure 4A and the write cycle of Figure 4B. As an example consider the situation of Figure 5, wherein the compare flip-flop is connected to two cells, *A* and *B*. If, after the first clock pulse of a cycle one cell receives a write signal then when the system returns to phase I, it has the same information as is stored in the compare flip-flop. The other cell sees only a compare cycle and returns to its initial state.

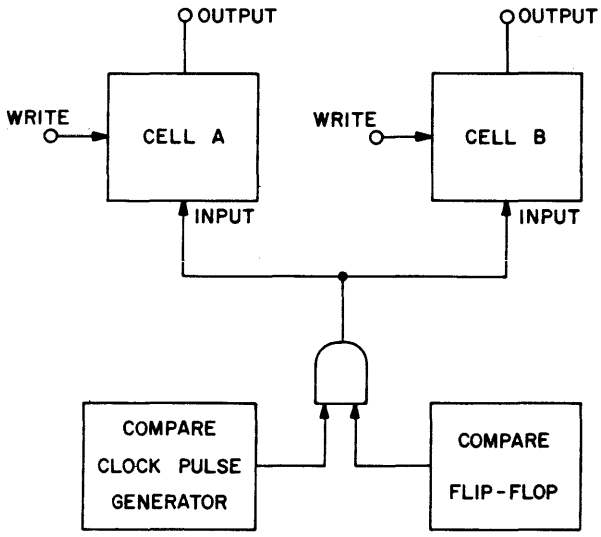


Figure 5. Two Associative Cells with a Common Compare Flip-Flop and Clock.

Although this is not the only method of writing into associative cells, other techniques will not be discussed here.

The "Don't Care" Comparison

As mentioned above, a cell should be able to respond to a "don't care" comparison so as to yield a true output in phase II regardless of its information content in phase I. In doing so, it must not lose its information.

For the cell shown in Figure 2, additional output circuitry is required to provide this feature. One circuit which may be added is shown in Figure 6. The output terminal of the circuit of Figure 2 is connected to point 1 of Figure 6.

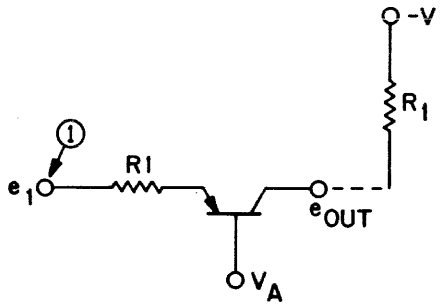


Figure 6. Cell Output Circuitry which Implements the "Don't Care" Logic.

The new output of the cell is the output of this network. The "don't care" feature is enabled through control of the voltage V_A.

When the output signal of the associative cell is a function of the state of the cell, V_A is at a negative voltage (say −2.0 volts). When $Q1$ is in saturation, e_1 is close to ground and an emitter current flows in $Q3$. Current is available at the collector of $Q3$ if required. If the current is sufficient to supply that required by R_1 then $Q3$ can saturate and e_{out} is about −2.0 volts. This is a false output. When $Q1$ is cutoff, e_1 is more negative than −2.0 volts so that $Q3$ is cutoff and no current is available at the output. This is a true output. The "Don't Care" comparison is made when V_A is at ground instead of −2.0 volts. Since $Q3$ is cutoff regardless of the condition of $Q1$, e_{out} is always true.

If the "don't care" control is separate from the 1 or 0 search control, it is possible to operate on the flip-flop as though searching for a 0 or 1 while masking this operation with "don't care."

The interconnection between a cell with the "don't care" feature and the full compare cell controlling it is shown in Figure 7. A full compare cell consists of both the input and the V_A controls. Each of these controls contains a memory element and output logic.

The associative cell's input controls, those determining whether or not the associative cell complements on the generation of a compare clock pulse, are the "AND" gate and the compare flip-flop shown previously.

The V_A controls are a switching amplifier and a memory element. The amplifier holds V_A either at ground or at the appropriate negative voltage. V_A is at the negative voltage when the V_A flip-flop contains a 0, and V_A is at ground when it contains a 1.

To summarize: A compare cell may be in any one of four different states. The compare flip-flop may either permit or inhibit complementing of the flip-flop by the clock and the V_A flip-flop may cause V_A to be at ground or a negative voltage. Each of these four states of the compare cell is designated by one of the following four symbols; 0, Ø, 1 and 1̸. These symbols are defined as follows:

0: "Zero" or "Zero Compare." The compare flip-flop is set so as to inhibit a compare clock pulse. The V_A control flip-flop contains a 0 (allowing a normal readout from the associative cell).

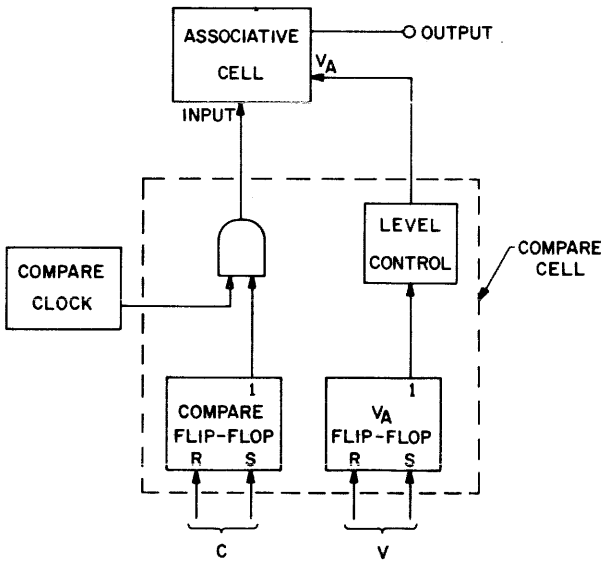


Figure 7. Associative Cell, Compare Cell and Clock with Logical Interconnections.

Ø: "Masked zero." The compare flip-flop is set so as to inhibit a compare clock pulse. The V_A control flip-flop contains a 1 (causing V_A to be at ground).

1: "One" or "One Compare." The compare flip-flop is set so as to permit a compare clock pulse to complement the associative cell. The V_A control flip-flop contains a 0.

ɣ: "Masked One." The compare flip-flop is set so as to permit a compare clock pulse to complement the associative cell. The V_A control flip-flop contains a 1.

Figure 8 is a Truth Table representation of the associative cell's output as a function of the state of the compare cell and its own information content.

There are times when a second gated output of the type shown in Figure 6 is added to an associative cell. The second output is "gated" (as the first one is by the voltage V_A) by a separate set of controls. For instance, in a group of associative cells storing a word of information, the second output of these cells might be used for word addressed readout. The control voltage of the second output, which is called V_B to distinguish it from control voltage of the associative output, would be common for all bits in a given word. The outputs would be individually connected to appropriate bit positions of a readout register.

A flip-flop with the two separately controlled outputs is shown in Figure 9. The output transistor $Q3$ is connected to the voltage V_A which is controlled by the compare cell. The output transistor $Q4$ is connected to V_B. $Q3$ conducts when it is enabled and when the flip-flop is in the 1 state. $Q4$ conducts when it is enabled and the flip-flop is in the 0 state.

STATE OF COMPARE CELL	INITIAL STATE OF ASSOCIATIVE CELL	OUTPUT OF ASSOCIATIVE CELL PHASE I	PHASE II
0	0	T	T
0	1	F	F
Ø	0	T	T
Ø	1	T	T
1	0	T	F
1	1	F	T
ɣ	0	T	T
ɣ	1	T	T

Figure 8. An Associative Cell's Output Signal for All States of a Compare Cell.

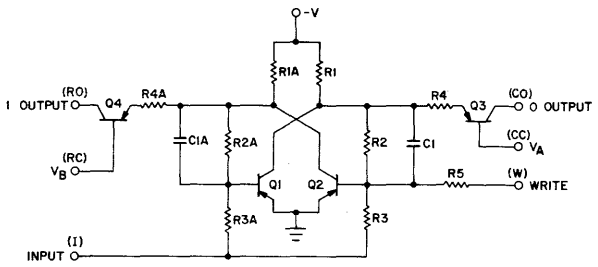


Figure 9. A Cell with Two Controlled Outputs.

Figure 10 is a black box which represents the circuit shown in Figure 9. The various input and output labels have the following significance:

I: Complementing input. A pulse applied here complements the flip-flop.

W: Write line. A negative signal on this line sets the cell to the 0 state, a positive pulse sets it to 1.

CO: Compare output. Produces the appropriate associative output signal when the system is in phase II.

CC: Compare control. V_A is connected to this point. A true CC input enables the 1 or 0 comparison from CO.

RO: The read output. This output is true if the RC input is false or if the RC input is true and the flip-flop is in the 1 state.

RC: The read control input. V_B is connected to this point. A true RC input enables reading.

An Associative Word

The arrangement of a three-bit associative word, a three cell compare register and other circuitry is shown in Figure 11. The compare

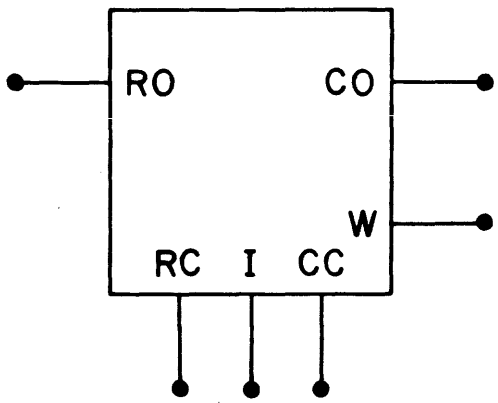


Figure 10. Black Box Representation of the Two Output Cell.

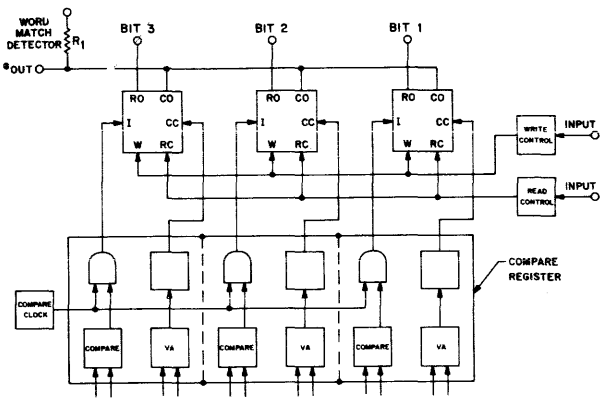


Figure 11. A Three Bit Associative Word with a Compare Register.

outputs of the associative word are commoned and have a single load resistor at the common point. The voltage e_{out} of the common point is at one of two values. If all the compare outputs of the word are true, then e_{out} is at about $-V$ volts (say -6 volts). If, however, one or more CO's are false then e_{out} is at around -2.0 volts (the value of V_A).

In the light of the preceding remarks, we can say that the CO's of the word are "ANDED." All CO's must be true for e_{out} to be true; e_{out} is the signal produced by the "Word Match Detector." If it is true during phase II then the associative word matches the word in the compare register. A mismatch in any bit position will cause e_{out} to be false.

The compare register may cause a word to match under several sets of conditions. To clarify this point consider the following example: An associative word contains the information 101 (read from bit 3 to bit 1). If the word 101 is loaded into the compare register and the system goes into phase II, then a match is indicated. However, since the "masking" operation causes an associative cell's output to be true regardless of its state, then the following words in the compare register also cause the associative word 101 to match: A) 10Ø, B) ØØØ, C) ØØØ, D) 111, E) 1Ø1, and 21 other combinations.

Comparisons made with compare words such as *A*, *B*, and *E* are called partial field comparisons. In such comparisons certain bits in the associative word must match those in the compare register, but in other bit positions, the associative cell may have either value. For an associative word to match, compare word *A*, for example, it must have a 1 in bit position 3, and a 0 in bit position 2 but may have either a 0 or a 1 in bit position 1. Associative words storing 101, 100, 000, or 001 would match compare word *B*. Compare words *C* or *D* cause any associative word to indicate a match regardless of its information content. Furthermore any word matches with the system in phase I as well as in phase II.

The read and write controls in Figure 11 are common to all bits in the associative word. The whole word may be written into or read out in a single cycle. When triggered, the write control sets all cells to 0. The read control, when triggered, enables the RO signal on all bits in the word. Interpreting the RO signal is not always a straightforward operation. While the system is in phase I, the RO signal is determined by the information content of the cell, true when the cell stores a 1, false when it stores a 0. However, in phase II the RO signal is a function of both the cell information and the contents of the related compare cell. If the compare cell stores a 0 then the RO signal is correct in phase II, but if the compare cell stores a 1, then the RO signal must be inverted.

AN ASSOCIATIVE MATRIX

A typical associative matrix is shown in Figure 12. It is made up of 3 associative words. Each word has 4 bits. It has a compare register, and a readout register. There is a word match detector (WMD), a read control and a write control for each word.

The one point shown in this matrix, which was not fully demonstrated in Figure 11, is the connection of the RO outputs to the readout register. The RO's of a given bit position make up a logical "AND" just as the CO's of a word do. All RO's must be true to produce a true signal at the readout register. If any one (or more) is false, the readout register will receive a false signal. Since a false signal can only be produced by a selected word, then the state of each bit in a selected word will be duplicated in the readout register.

The read control circuits are gated amplifiers. Figure 13A shows typical gating to one of these circuits. Either of two input gates may enable the circuit. The first gate has two "ANDED" inputs, one from an address register, the other from the 0 side of an address control flip-flop.

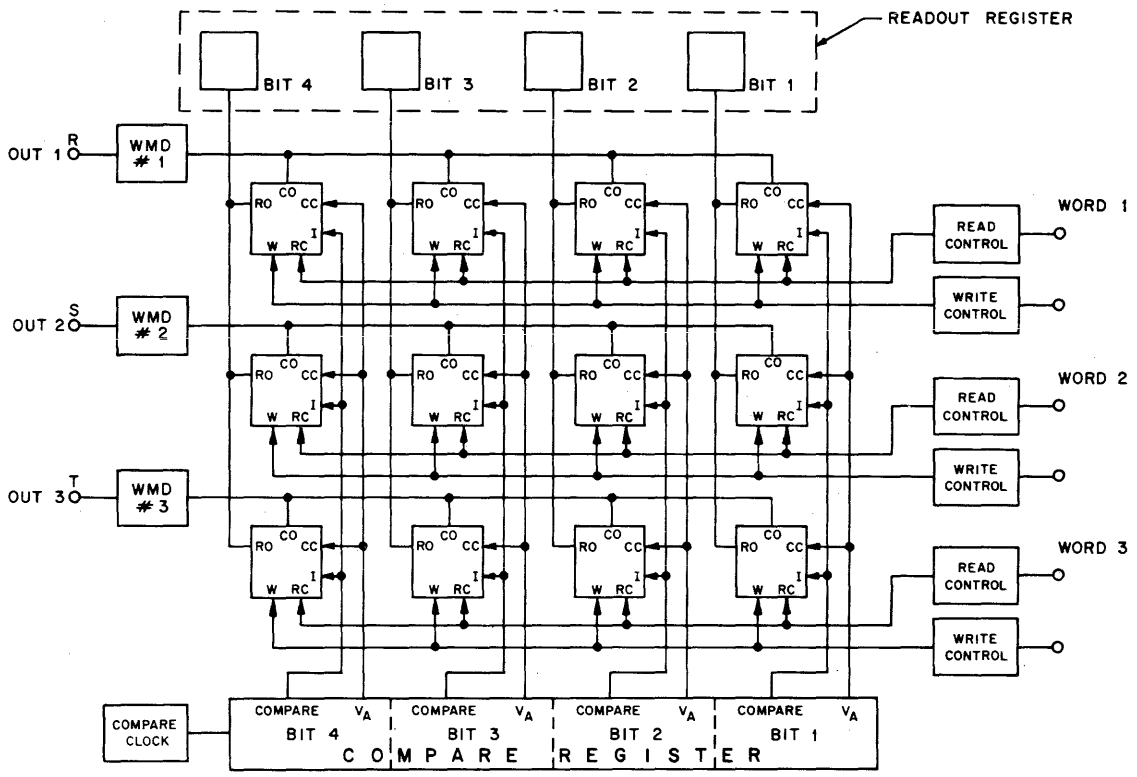


Figure 12. A Three Word Associative Matrix with a Compare Register.

The second gate also has two "ANDED" inputs, one from the word's own WMD and the other from the 1 side of address control flip-flop. The address control flip-flop state determines whether the read operation is to be performed on a matching word or on a word whose physical address is contained in the address register.

The word write control circuits are also gated amplifiers. As shown in Figure 13B, they have an input from the address register which is "ANDED" with a "write" clock.

Reading Words from the Matrix

As indicated above there are two addressing means for reading from an associative matrix. One is to read from a selected physical location, the other is to read from a word containing specified information in certain of its bit positions.

Reading from a particular physical location is called "location addressed reading." The other form of reading is called "content addressed reading." Content addressed reading may produce multiple responses. Then it is necessary to have a scheme whereby matching words may be accessed sequentially. A solution to this problem is described in the section on multiple matches. At this point we assume that only one word responds to either form of addressing.

The operating sequences necessary to perform each type of reading are described below step by step:

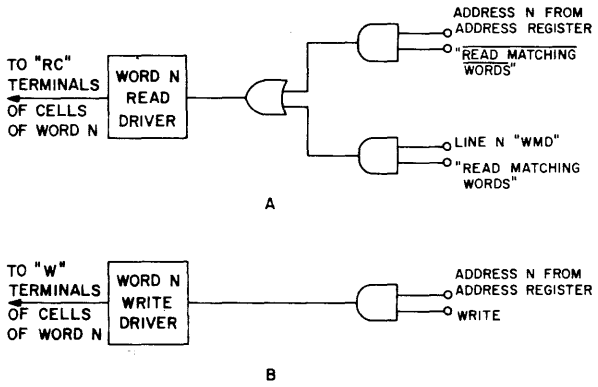


Figure 13. A Read and a Write Driver and Their Input Controls.

Location Addressed Read (LAR)

A. Load the address register with the physical address of the location to be read.
B. Set the address control flip-flop so as to enable word selection from the address register.
C. Strobe the readout register.

Content Addressed Read (CAR)

A. Enter identifying bits into the compare register. Enter Ø into all other bit positions of the compare register. Set the address control flip-flop to enable readout of a matching word.
B. Generate a compare clock pulse so as to put system into phase II.
C. Strobe the readout register.
D. Generate compare clock pulse so as to return system to phase I.

An Example of CAR

Assume that the associative matrix of Figure 12 contains the information 1100 in word 1, 1001 in word 2 and 0101 in word 3. When the problem is to find what other information is contained in any word which has a 0 in bit position 4 and a 1 in bit position 3, the following occurs:

A. Enter 0 1 Ø Ø in the compare register.
B. Enter phase II.
   At this point the states of the cells in the words and their associative output signals are these:

	Cell States	*CO Signals*
Word 1:	1000	FTTT
Word 2:	1101	FFTT
Word 3:	0001	TTTT

   Only word 3's WMD will be true and its read control circuit will enable the RO's of word 3.
C. Strobe readout register.
   The readout register will see 0001, but since the third bit of the compare word is known to be 1, then the word readout is modified to 0101.
D. Return system to phase I. The initial information content of all associative words is restored.

Writing Words into the Matrix

Writing may occur under one of three sets of conditions. A word may be written into a specified physical location. A word may be written into any convenient position which doesn't already contain information. A word may be written into whatever position a second, specified word occupies at the beginning of the write cycle. The first operation is called Location Addressed Writing (LAW), the second Loading, and the third Content Addressed Writing (CAW).

All three operations involve location addressed writing, but in Loading and CAW it is also necessary to find the location which meets the specified requirements. The latter two require a compare cycle previous to or concurrent with the location addressed writing operation.

Location Addresed Writing (LAW)

A. Enter into the compare register the information to be written. Enter into the address register the address of the location to be written in.
B. Generate the first compare clock pulse so as to send the system into phase II.
C. Generate a write pulse at the address selected by the address register.
D. Generate the second compare clock pulse so as to return the system to phase I.

Loading

Loading may follow one of two possible sequences depending on the manner in which an unused word location is identified. It can be identified by an extra bit in each word. This bit might be a 0 when no information is stored in the related word location and a 1 when information is stored. An unused word location can also be identified by the fact that it contains a 0 in every bit position during phase I. This of course means that the code word of all 0's cannot be used as an information word.

The second approach is more efficient in that for N bit positions 2^N-1 code words are usable. The first approach can only accommodate $2^{(N-1)}$ code words.

If the additional bit position is used then it must have a corresponding cell in the compare register. This cell will contain a Ø or a 1

when searching for a word. It will contain a 0 when looking for an empty word.

Load Sequence 1

(Wherein an extra bit identifies an empty word location)

A. Enter the word which is to be written in the matrix, into the compare register, but mask all bits. That is for 1's enter 1 and for 0's enter Ø.
   Set the extra bit compare cell to 0.
B. Sense the WMD's and enter address of true WMD into the address register.
C. Generate first compare clock pulse so as to send the system into phase II.
D. Generate a write pulse at the address selected by the address register.
E. Return the system to phase I with a compare clock pulse.

Load Sequence 2

(Wherein an empty word location contains all 0's)

A. Enter the word which is to be written in the matrix, into the compare register.
B. Sense WMD and enter address of true WMD into the address register.
C. Generate the first compare clock pulse.
D. Generate a write pulse at the address determined by the address register.
E. Return the system to phase I.

In both of the above sequences, the WMD is sensed while the system is in phase I. As long as no 1's are being used in a comparison, it is not necessary to go to phase II for sensing. In sequence 1, only the extra bit value counts and a 0 is compared there while all other bits in the compare register are 1 or Ø. In sequence 2, all 0's are to be compared when searching for an empty word location. A word storing all 0's will always have a true WMD in phase I.

Content Addressed Writing (CAW)

A. Enter the identifying bits of the word which is to be altered into the compare register.
B. Send the system into phase II.
C. Sense the true WMD and enter its address into the address register
D. Return system to phase I.
E. Put in the compare register the word to be written in place of the old word.
F. Send system into phase II.
G. Generate write pulse at address designated by the address register.
H. Return system to phase I.

An Example of CAW

Assume that the associative matrix of Figure 12 contains 0101 in word 1, 0110 in word 2 and 1111 in word 3. The word 0001 is to be written in the location in which 0101 is stored.

A. 0101 is entered into the compare register.
B. The system enters phase II so that word 1 goes to 0000.
C. The WMD of word 1 is true so that a "1" is stored in the address register.
D. The system returns to phase I. The information content of the word is unaltered.
E. 0001 is entered into the compare register.
F. The system is sent to phase II.
G. A write pulse is generated on word 1.
H. The system returns to phase I. Word 1 contains 0001, word 2 contains 0110 and word 3 contains 1111.

A Special Case of CAW

There may well occur in associative matrices the situation in which only a portion of a word would be altered and the fixed portion of the word would be used to address the system. Then this sequence could be used:

A. Enter full word into the compare register, but mask those bits not used for addressing.
B. Send the system into phase II.
C. Sense the WMD and enter the address of the matching word into the address register.
D. Generate a write pulse on the word designated by the address register.
E. Return the system to phase I.

An Example of the Special Case of CAW

Assume that the associative matrix of Figure 12 contains 0101 in word 1, 1001 in word 2 and 1111 in word 3. The word with 01 in bits 4 and 3 is to have its last two bits altered to 10.

A. Enter 01 ɪ Ø into the compare register.
B. Send system to phase II. Only the WMD of word 1 will be true since the other words will have at least one associative mismatch in bit positions 3 and 4.
C. Since the WMD of word 1 is true, the address "1" is entered into the address register.
D. A write pulse is generated on word 1.
E. The system returns to phase I. Word 1 contains 0110, word 2 contains 1001, word 3 contains 1111.

Multiple Matches—in Two Dimensions

In discussing the reading and writing sequences we have assumed that any compare operation results in a unique associative response. This is not always the case, especially when the load operation is performed. It is necessary to be able to choose one address at a time when many respond to a content comparison. This problem is simply solved in a two dimensional system by introducing priority gating. When two or more responses are generated, they are equally valid. That is, if the locations which respond are to be read sequentially or written into sequentially, it doesn't matter which location is chosen first. If it did matter, then the operator must have failed to label his word adequately prior to comparison since it must be some additional information contained in one word which makes it preferable to others. Since the matching words have equal validity, we may sequentially output responding words in an arbitrary order. A priority network which makes the selection on the basis of the relative physical locations of the words is described below. This priority network is shown in Figure 14. It consists of memory elements and gating. Each memory element is associated with a specific word location.

Assume that all three WMD's are true when the "Sense WMD" command occurs. Then, all three flip-flops are set, their S outputs are true and their $\overline{S}$ outputs false. Then only the "AND" gate 203 (of the group 203, 204 and 205) is true. Since gate 203 controls the word 1 read and write circuits (See Figure 13A and 13B) then they alone may be true. If, after the appropriate read or write operations have occurred, a clock pulse occurs on line 301, then

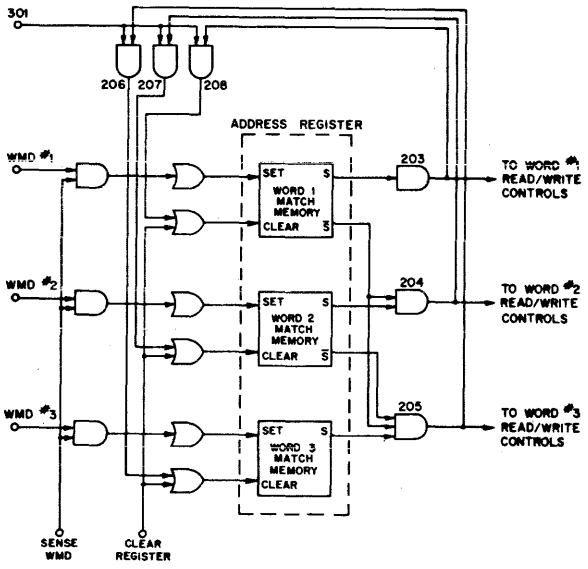


Figure 14. Priority Gating.

the "word 1 match memory" will be cleared by gate 208. With this cell cleared gate 203 will be false and gate 204 will be true. Word 2 may be operated on. Similarly, after operating on word 2, line 301 may be pulsed again, clearing the word 2 match memory. Gate 204 will then be false and 205 will be true allowing operation on word 3. Again, if WMD 1 and 3 are true, so that only memories 1 and 3 are set, word 1 will be operated on first followed by word 3.

The memory elements are not strictly necessary in the priority networks. We could simply produce the "not" function of each WMD, and make up gate 203, 204 and 205 directly. However, this technique leaves the problem of how to read out more than the highest priority word.

Solving this problem requires a method whereby a word, once it has been operated on, will be caused to mismatch—without changing the information content of the word. This can be accomplished by adding an extra associative bit to every word. This bit is not the same one used for load sequence 1. This extra bit contains a 0 at the beginning of every operation. When a word is operated on, read or write, the extra bit is set to 1. The compare register always searches for 0 in this bit position so that setting the bit to 1 induces a mismatch in the word. The additional bit in every word somewhat offsets the elimination of the memory cells in Figure 14.

Three Dimensional Associative Matrices

The usual arrangement of associative matrices with over 100 words will probably be three dimensional. That is, a word's physical location will be designated by, and arranged in, two coordinates and its bits in a third coordinate. This is identical to the arrangement of cores in a random access memory. The word will be addressed by its x, y coordinates. The bits of a word will lie along the z axis, (see Figure 15).

The reason for this address organization is similar to that for the normal memory. It results in a reduction of addressing and sensing equipment. However, in the associative matrix the reduction may not be comparable to that in a core memory since, for some operations, coincident addressing is not adequate.

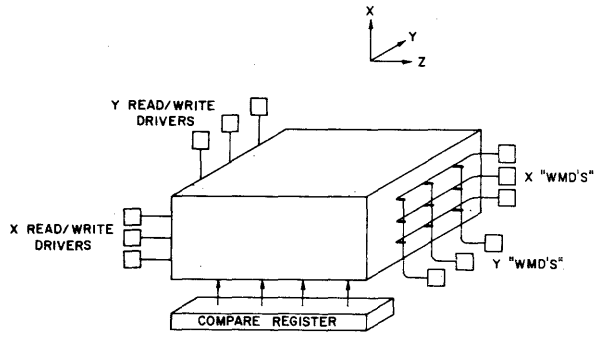


Figure 15. Three Dimensional Organization of an Associative Memory.

Figures 12, 13, and 14 display some circuitry common to the three dimensional matrices as well as to the two dimensional ones. The compare register is connected to all words as it is in Figure 12. The WMD's are arranged in a different manner.

The connections from the x and y word match detectors to the word outputs is shown in Figure 16. Each word output goes to two WMD's, one which represents its x address and one which represents its y address. When a word matches both WMD's to which it is connected go true. The outputs of all words on an x or y plane are "OR'd" to the plane's WMD so that if one or more words on a plane match, the WMD goes true.

The output of a WMD is logically connected to the read and write drivers of its planes. In order to read from or write into a particular word both the X and the Y read or write driver associated with it must be on.

All the basic read and write sequences are the same for the three dimensional arrangement as for the two dimensional case. The only additional consideration is that addresses are now given, and selected, in terms of an X and a Y location. The usual coincident access techniques must be employed to implement this.

Multiple Matches in a Three Dimensional Matrix

The problem of resolving multiple matches is more complicated in the three dimensional case. It is not always possible to solve the prob-

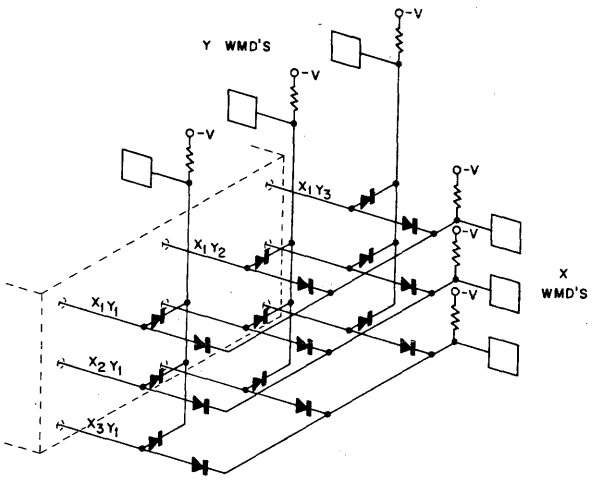


Figure 16. Organization of the WMD's in the Three Dimensional Matrix.

lem with priority gating. Two examples will illustrate this.

In the following two examples, there is a 9 word associative matrix arranged in 3 X planes, labeled X_1, X_2, and X_3 and 3 Y planes labeled Y_1, Y_2, and Y_3. Any word in the matrix has a unique XY location.

Case 1: A CAR sequence is started. At compare time four locations match, X_1Y_3, X_2Y_1, X_2Y_2 and X_3Y_2. Consequently, WMD's X_1, X_2, X_3, Y_1, Y_2 and Y_3 are true. It is not possible to select any pair of true X and Y WMD's and wind up addressing one of the matching words. That is, WMD's X_1 and Y_1 are both true, but address X_1Y_1 is not one which

corresponds to a word which matched. All that is known is that there is at least one word in the X_1 plane which matches. The Y address(es) of the word or words is unknown.

Case 2: A CAR sequence is started. At compare time three word locations match: X_1Y_1, X_1Y_2, and X_1Y_3 so that WMD's X_1, Y_1, Y_2 and Y_3 are true. It is possible to resolve this multiple match by simple priority gating on the Y WMD outputs. This is so because only one X WMD responded.

The two cases both illustrate the problem and indicate a solution to it. (Another method for resolving the problem has been suggested by Frei and Goldberg in reference 15.) Any set of multiple matches involving only one X or Y plane can be resolved by priority gating on the other plane. Therefore, when a set of matches involves multiple X and multiple Y planes, an operation must be performed which will prevent matching signals from all words but those on one of the responding X or Y planes. Then, the match or matches produced by the words on this plane can be sequentially operated on through priority gating on the WMD outputs of the other plane. We will cover one method of implementing this operation.

Two additional bits are required in each associative word, these bits, and their drivers are shown in Figure 17. The bits in column A are in the 0 state at the beginning of a cycle. They are set to 1 by a coincidence of a true read control line on the word, and a "Read Mark" clock. They are identical to the extra bits suggested in the preceding section. The compare register cell for column A normally contains 0. The bits in column A simply cause a word to mismatch once it has been read. The bits in column B are in the 1 state at the beginning of a cycle. They are set to 0 when the appropriate "X plane column B" control is triggered. The compare cell B normally contains ∅. When the X plane control is triggered the column B bits in words X_1Y_1, X_1Y_2 and X_1Y_3 are all set to 0.

Reconsider Case 1 for a system which includes the A and B bit planes shown in Figure

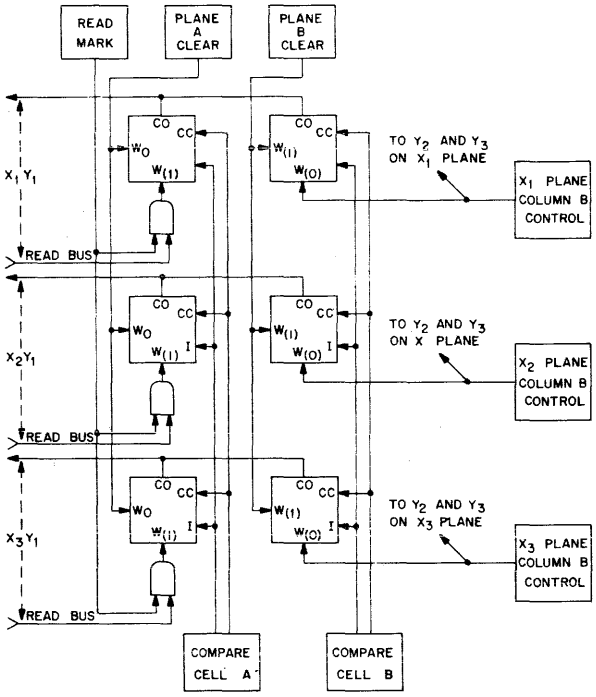


Figure 17. Control and Organization of Special Bits.

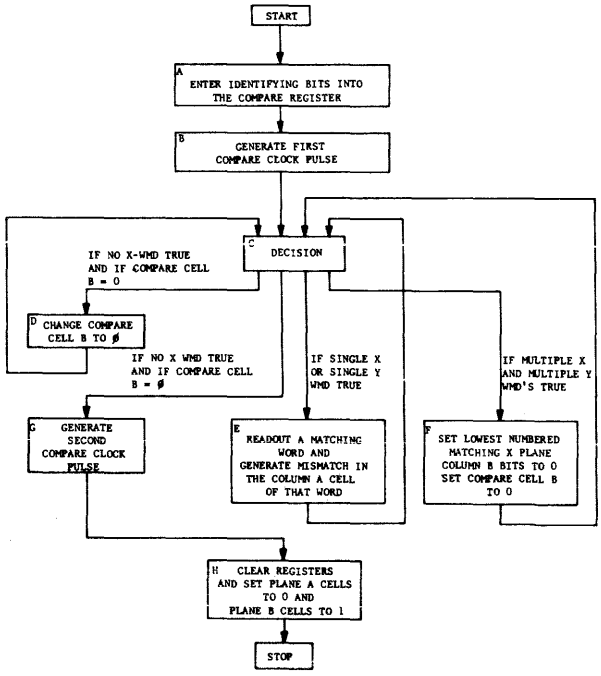


Figure 18. CAR Operation in which Multiple Matches are Readout Sequentially.

17 and priority gating on the WMD outputs of the Y planes. There is no need for flip-flops on these outputs. The generalized timing diagram for resolving multiple matches in a CAR sequence is shown in Figure 18.

After the first compare clock pulse, words X_1Y_3, X_2Y_1, X_2Y_2 and X_3Y_2 match. Since there are multiple X and Y responses, the X_1 plane is isolated so that only X_1Y_3 matches. It is readout and caused to mismatch. Then the X_2 plane is allowed to match and X_2Y_1 and X_2Y_2 are readout in that order based on the Y priority gating. Finally X_3Y_2 is readout.

For other operations CAW, Loading and the like, the sequence for resolving multiple matches is similar to that shown in Figure 18. However, another operation or series of operations would be substituted for step E.

To implement the content addressed write (CAW) sequence, memory cells are added to the word match detectors. When step E is reached, instead of operating on the location indicated by the WMD's, the address of location is stored in the WMD memory cells. The system returns to phase I at which time the word to be written is entered in the compare register. The system goes into phase II and the new word is written in the location indicated by the WMD memory cells.

CONCLUSIONS

The investigation of solid state circuits used as associative cells should be started in earnest. The cells described are far from being the simplest cells, either logically or circuit-wise. We have designed and built solid state cells less than half as complex as those described.

The solid state cell offers several technically attractive features. Its output signal to noise ratio is on the order of 10^6. This is several orders of magnitude greater than that provided by magnetic cells, such as bi-cores or transfluxors. The signal to noise ratio is a principle factor in the possible length of associative words.

The solid state cell has input and output signal energy levels which are roughly equivalent. Thus cross talk problems, a real headache in large arrays, are minimized.

The input energy requirement of a solid state cell can be made many orders of magnitude lower than for magnetic cells. The input requirements are the principle limiting factor for the possible number of words in an associative memory.

The output signals of the solid-state cell are compatible with the external logic. This eliminates the need for buffer circuitry such as sense amplifiers. Also these signals can be maintained for as long as they are needed.

The component tolerances in the solid state cell are very loose. Transistors require $h_{FE's}$ of as low as 15, low breakdown voltages, etc. Resistors may have tolerances of $\pm 30\%$, capacitors may have tolerances of $\pm 50\%$. Loose tolerance requirements are a boon to integrated circuits.

The solid state circuit can be fabricated today from standard components and used in a normal environment. For a price, large memory arrays could be built today with standard components. If the 2N404 were used along with loose tolerance capacitors and resistors, the component cost of the cells described here is less than $2.00 in every case. We have designed cells performing the most complex function outlined here whose component cost, today, approaches $1.00.

We have constructed and operated at a 2.5 mc clock rate a 9 word (8 bits per word) Associative Memory. This memory performs all operations described here and in addition can sort information using a routine not yet described in the literature.

As promising as word organized associative matrices look, thinking should not be confined to applying associative cells in this manner. For instance, the associative cell seems to be a natural in iterative logical operations. To properly investigate these possibilities, more logicians should be brought into the picture.

BIBLIOGRAPHY

1. A. E. SLADE and H. O. MCMAHON, "A Cryotron Catalog Memory System," Proceedings of the Eastern Joint Computer Conference, December 1956.
2. A. E. SLADE and C. R. SMALLMAN, "Thin Film Cryotron Catalog Memory," Sympo-

sium on Superconductive Techniques for Computing Systems, May 1960.

3. R. R. SEEBER, "Cryogenic Associative Memory," National Conference of the Association for Computing Machinery, August 1960.
4. P. DAVIES, "A Superconductive Associative Memory," Proceedings Spring Joint Computer Conference, 1962, p. 79.
5. V. NEWHOUSE, "A Cryogenic Data Addressed Memory," Proceedings Spring Joint Computer Conference, 1962, p. 89.
6. R. ROSIN, "An Organization of an Associative Cryogenic Computer," Proceedings Spring Joint Computer Conference, 1962, p. 203.
7. M. ASHER, "G. E. Cryogenic Associative Memory Circuit Developed," Electronic News, March 19, 1962, p. 59.
8. R. C. MINNICK, "Magnetic Comparators and Code Converters," Symposium on the Application of Switching Theory in Space Technology, February 1962.
9. J. R. KISEDA, H. E. PETERSON, W. C. SEELBACH and M. TEIG, "A Magnetic Associative Memory," IBM Journal, Vol. 5, #2, p. 106, 1961.
10. W. MCDERMID, "A Magnetic Associative Memory System," IBM Journal, Vol. 5, #1, January 1961, p. 59.
11. R. J. KOERNER, "Memory Array Searching System," U. S. Patent #3,031,650.
12. ALAN CORNERETTO, "3-K Bit Associative Memory Works at Room Temperature," Electronic Design, July 5, 1962, p. 8.
13. W. K. ORR, "Look Ahead Logic Simplified," to be published.
14. R. R. SEEBER, "Associative Self-Sorting Memory," Eastern Joint Computer Conference, December 1960.
15. E. H. FREI and J. GOLDBERG, "A Method for Resolving Multiple Responses in a Parallel Search File," IRE Transactions on Electronic Computers, December 1961, p. 718.
16. R. R. SEEBER and A. B. LINDQUIST, "Associative Memory with Ordered Retrieval," IBM Journal, January 1962, Vol. 6, #1, p. 126.
17. MORTON H. LEWIN, "Retrieval of Ordered Lists from a Content Addressed Memory," R.C.A. Review, June 1962, p. 215.
18. J. ATKIN and N. B. MARPLE, "Information Processing by Data Interrogation," IRE Transactions on Electronic Computers, Vol. EC-11, #2, April 1962.

ACKNOWLEDGEMENTS

Many thanks to Arnold Jorgensen and Larry Bewley who provided the opportunities and the encouragement this work required; also to Jean Holtman who typed it in its many forms, and Maurine Yenglin for her fine illustrations.

PROGRAMMING AND DESIGN CONSIDERATIONS OF A HIGHLY PARALLEL COMPUTER*

Jon S. Squire
Sandra M. Palais
Information Systems Laboratory
University of Michigan
Ann Arbor, Michigan

A number of automata and automatic computing devices have been proposed whose operations could be considered highly parallel. These include von Neumann's tesselation model for self-reproducing automata,[6] the ENIAC computer,[1] John Holland's iterative circuit computer,[3] S. H. Unger's spatially oriented computer,[10] and the SOLOMON computer.[8] The literature available on these machines indicates programming facility played a subordinate role to design. Cf. Newell,[5] Schwartz,[7] Garner and Squire.[2]

The purpose of this paper is to present a machine organization where the emphasis is on the programmability of highly parallel numerical computation. Ease, convenience, and economy of design are secondary considerations.

A highly parallel computer is defined as a machine that is capable of simultaneous execution of instructions where computation time is limited by numerical analysis rather than lack of hardware. Since the execution of an instruction depends on data, over which the programmer has no control, the hardware provides the necessary synchronization. There is no restriction on the location of data or instructions because algorithms can require an array of instructions to execute simultaneously on scattered groups of data (for example, the test for convergence of a number of independent procedures).

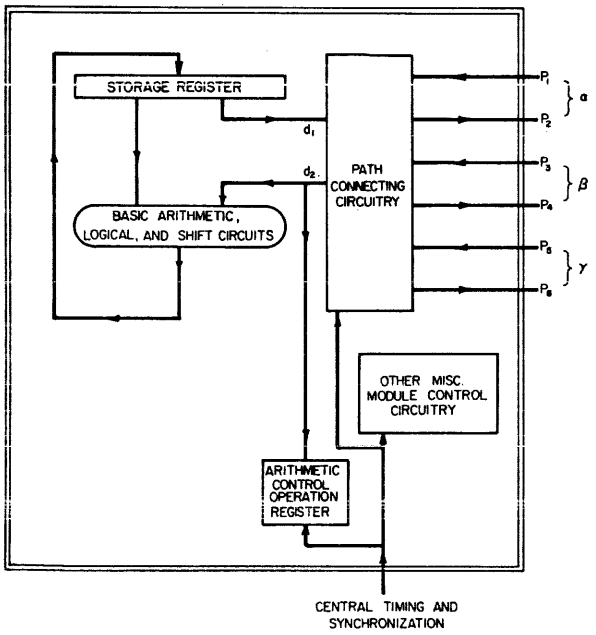
Some specific examples will show how programs can be written where many of the instructions can be executed simultaneously during an execution cycle. Moreover, the number of execution cycles is far less than would be required on a single processor computer. Consider first the simple task of summing N numbers. On a conventional computer this requires N-1 execution cycles, while on a highly parallel computer only $1+log_2N$ execution cycles are required, e.g. given 1000 numbers perform 500 additions (on disjoint pairs) during the first execution cycle, then perform 250 additions on pairs of these results during the second execution cycle, etc. Thus, there is a ratio of 999 to 11 in the number of execution cycles. Next, consider multiplying a 20 element vector, B, by a 20 $\times$ 20 matrix, A. On a single processor computer, 780 additions and multiplications are required to compute $c_i = \sum_{j=1}^{20} a_{ij}b_j$ for j=1,20. On a highly parallel computer, the 400 multiplications are performed during the first execution cycle, and only six more execution cycles

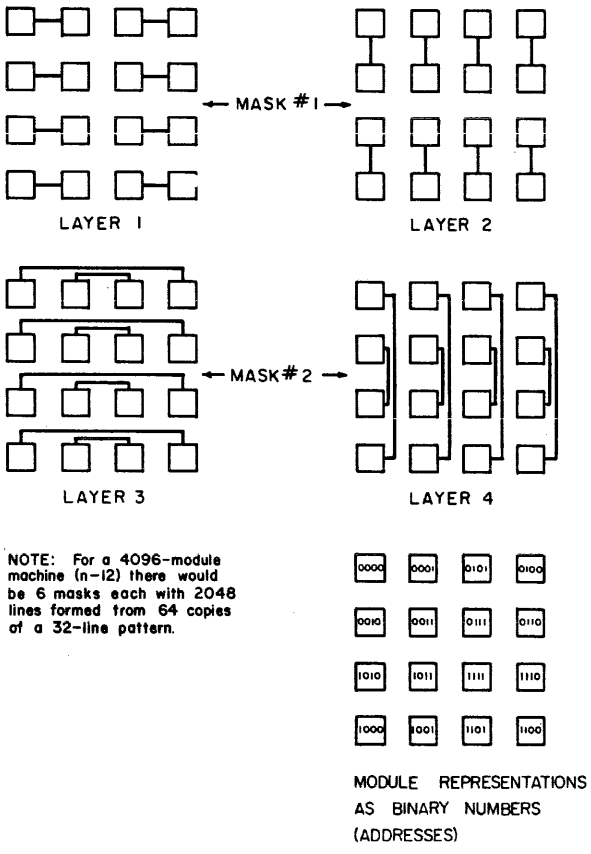
* Under contract with the United States Air Force, Aeronautical Systems Division, Contract No. AF 33(657)-7391. Wright-Patterson Air Force Base, Ohio.

are needed to produce the 20 sums of 20 numbers each. Thus, a ratio of 780 to 7 in the number of execution cycles is obtained. Other computations studied in detail include matrix inversion where several matrices of instructions are used to obtain a ratio of 40,000 to 61 in the number of execution cycles. The algorithm used is given in,[4] and the highly parallel computer program is given in Appendix A. And finally, in demonstrating programmability, a ratio of over 50 to 1 is obtained in the number of execution cycles on the numerical solution of 20 second-order differential equations using the fourth-order Runge-Kutta method.

A formal procedure was also devised for detecting the degree of parallel computation obtainable from a given algorithm. This work is presently being prepared for publication.

The following computer organization is presented to show that the design of a highly parallel computer is technically feasible. The computer is constructed from a number of identical modules. Each module contains: 1) one word of storage which may contain either an instruction or data, 2) arithmetic and operation decoding circuits, and 3) path-connecting circuitry. See Fig. 1. There are 2^n modules (words of storage) each directly connected to n other modules. In particular, each module is placed at the vertex of an n-dimensional cube, and the edges of the n-cube are wires. See Fig. 2 for the two-dimensional layout of the machine. Execution proceeds as follows: (a) Each module containing an instruction to be executed becomes active. (b) Paths are formed from active modules to modules designated by the three addresses of each instruction. A path may enter a module, pass through a gate in the path-connecting circuitry, and leave a module without affecting the storage or arithmetic of the module. (c) Operands are routed along paths to the modules where arithmetic operations are performed. (d) The operations are performed. (e) When all operations are completed and paths removed, the next execution cycle begins.





The motivation in choosing this organization for obtaining the desired programming ability stems from the following ideas about realizing a highly parallel computer. 1) The only way to be sure there are enough operation decoders and arithmetic units available is to have one for each word of storage. This also avoids all priority problems associated with assigning instructions to processors. Further, with more than one word of storage per module, there

would be insufficient path connecting to give simultaneous random access. 2) The modules could be identical, thus interchangeable for servicing, and adaptable to mass production techniques. 3) The n-cube is the geometric configuration providing maximum interconnection while retaining one-half the vertices isolated from each other. Without such isolation there is the unsolvable priority problem of two identical modules attempting to connect to each other simultaneously. In this case, either both connect and computation errors occur, or neither connect and the machine hangs up. The isolation is accomplished by having all modules with an even number of ones in their address extend paths one unit. Then, the remaining modules extend their paths one unit, etc. The logic for determining which of a module's n neighbors a path should connect to is performed by a simple logical circuit. The address of the path's destination is ring-summed with the module's address. Each bit position where a one results designates a module that is one unit of Hamming distance closer to the destination. The only remaining priority problem is when two paths emanating from the same module require the same path segment. Since this is not a function of the state of any other module, a simple logical matrix can be used to allow one path to extend and reroute the other path. Further details about operation codes, programming and logical design are given in the Information Systems Laboratory Report.[9]

The number of simple logical components required by a 4096-module machine as described above is approximately 5 million. A partial breakdown would include 1.1 million components for storage, programmable and internally used; 1.6 million components, each, for path connecting and arithmetic circuitry; and another .5 million components for miscellaneous control, including central timing and synchronization. The building of a 4096-module machine would require about 20 times as many components as STRETCH and, as such, is within engineering feasibility. The economics would then depend on an application needing a speed advantage of 100 to 1 or more, and finally on the abilities of logical and circuit designers and component manufacturers.

The organization of a highly parallel computer has been proposed from the programming point of view. As such, it fulfills the implications of its name by being easily utilized for parallel computation. It has been shown that hundreds of processors are needed to decode and perform hundreds of operations simultaneously. It has also been shown that simultaneous random access is required in order for the arithmetic circuits to fetch their operands. In these ways, no lesser organization could accomplish the same task.

APPENDIX A

MATRIX INVERSION PROGRAM FOR THE DESCRIBED MACHINE

INSTRUCTION LOCATION	OPERATION	ADDRESSES α,β,γ
*		
*		THE NXN MATRIX IN THE 'A' REGION IS
*		INVERTED BY A GAUSS-JORDAN METHOD.
*		THE INVERTED MATRIX REPLACES THE
*		ORIGINAL CONTENTS OF THE 'A' REGION
*		
ENTRY	PROCEED=	0,0,SET1'(1)
*		
*		FIRST EXECUTION STEP OF A THREE-EXECUTION
*		STEP LOOP THAT WILL BE PERFORMED N TIMES.
*		
SET1'(1)...SET1'(N)	INDADR	LA'(1),E(1),Q'(1)...LA'(N),E(N),Q'(N)
SET1'(N+1)	INDADR	EXIT
*		
E(1)...E(N)	LOAD	AKK,A(1,1),F(1)...AKK,A(N,N),F(N)
*		
LA'(1)...LA'(N)	INDADR	L(1,1)...L(N,1),...,L(1,N)...L(N,N)
L(1,1)...L(1,N)	LOAD	B'(1),A(1,1),M(1,1)...B'(1),A(1,N),M(N,1)
L(N,1)...L(N,N)	...	B'(N),A(N,1),M(1,N)...B'(N),A(N,N),M(N,N)
B'(1)...B'(N)	INDADR	AT(1,1)...AT(1,N),...,AT(N,1)...AT(N,N)
*		
Q'(1)...Q'(N)	INDADR	U(1,2)...U(1,N),,U(I,1)...U(I,I-1),U(I,I+1)...
	ETC	U(I,N),,U(N,1)...U(N,N-1)
U(1,1)...U(1,N)	DIVIDE	A(1,1),A(1,1)...A(1,N),A(1,1),T(1)
U(N,1)...U(N,N)	...	A(N,1),A(N,N)...A(N,N),A(N,N),T(N)
*		
*		SECOND EXECUTION STEP
*		
F(1)...F(N)	DIVIDE	A(1,1),AKK,G(1)...A(N,N),AKK,G(N)
*		
M(1,1)...M(1,N)	MULTIPLY	C'(1),A(1,1),S(1,1)...C'(N),A(1,N),S(1,N)
M(N,1)...M(N,N)	...	C'(1),A(N,1),S(N,1)...C'(N),A(N,N),S(N,N)
C'(1)...C'(N)	INDADR	AT(1,1)...AT(N,1),...,AT(1,N)...AT(N,N)
*		
T(1)...T(N)	LOAD	V'(1),ZERO,Y(1)...V'(N),ZERO,Y(N)
V'(1)...V'(N)	INDADR	A(2,1)...A(N,1),,A(1,I)...A(I-1,I),A(I+1,I)...
	ETC	A(N,I),,A(1,N)...A(N-1,N)

INSTRUCTION LOCATION	OPERATION	ADDRESSES α,β,γ
*		
*		
*		THIRD EXECUTION STEP
*		
G(1)...G(N)	DIVIDE	A(1,1),AKK,SET1'(1+1)...A(N,N),AKK,SET1'(N+1)
*		
S(1,1)...S(1,N)	SUBTRACT	A(1,1),AT(1,1)...A(N,1),AT(N,1)
S(N,1)...S(N,N)	...	A(N,1),AT(N,1)...A(N,N),AT(N,N)
*		
*		ON THE ITH PASS THROUGH THE LOOP
*		THE ITH ROW OF SUBTRACT INSTRUCTIONS
*		IS INHIBITED.
*		
Y(1)...Y(N)	INHIBIT	-,-,Z'(1)...-,-,Z'(N)
*		
Z'(1)...Z'(N)	INDADR	S(1,1)...S(1,N),...,S(N,1)...S(N,N)
*		
*		END OF COMPUTATION LOOP
*		
*		STORAGE ASSIGNMENT
*		
A(1,1)...A(1,N)	DATA	
A(N,1)...A(N,N)	...	
*		
AT(1,1)...AT(1,N)	TMPSTR	
AT(N,1)...AT(N,N)	...	
*		
AKK	TMPSTR	
*		
ZERO	DEC	0
*		
	END	

A few comments should allow the interested reader to understand the details of the matrix inversion program. 2+1 addressing is used where the first address is one operand and the location of the result. The second address is the other operand, and the third address designates the location of the next instruction. A prime indicates indirect addressing, and the pseudo operation INDADR sets up a tree of indirect addresses.

The sequence of instructions which form the "loop" is:

ENTRY — SET 1′ (1) → E(1) → F(1) → G(1)
— SET 1′ (2) → E(2) → F(2) . . . E(n) → G(n)
— SET 1′ (N + 1) → EXIT.

REFERENCES

1. "ENIAC, The Electronic Numerical Integrator," GOLDSTINE, A. and GOLDSTINE, H. H., *Math Tables and Other Aids to Computation*, Vol. I, pp. 97-110, July, 1946.
2. GARNER, H. L. and SQUIRE, J. S., "Iterative Circuit Computers," USAF-Westinghouse Workshop on Parallel Computers, Baltimore, Md., October 3-4, 1962. (Papers to be published in book form by the Permagon Press.)
3. HOLLAND, J. H., "Iterative Circuit Computers," *Proceedings of the 1960 WJCC*, pp. 259-265.
4. INGERMAN, P. Z., "Algorithm 140, Matrix Inversion," *Communications of the ACM*, p. 556, November, 1962.
5. NEWELL, A., "On Programming a Highly Parallel Machine to be an Intelligent Technician," *Proceedings of the WJCC*, p. 267, May, 1960.
6. "Notes on John von Neumann's Cellular Self-Reproducing Automaton," edited by A. W. Burks.
7. SCHWARTZ, E. S., "An Automatic Sequencing Procedure with Application to Parallel Programming," *Journal of the ACM*, p. 153, October, 1961.
8. SLOTNICK, D. L., et al., "SOLOMON Computer," *Proceedings of the 1962 EJCC*, p. 97.
9. SQUIRE, J. S. and PALAIS, S. M., "Physical and Logical Design of a Highly Parallel Computer," Information Systems Laboratory Technical Report 04794-2-T, University of Michigan, October, 1962.
10. UNGER, S. H., "Computer Oriented Toward Spatial Problems," *Proceedings of the I.R.E.*, p. 1744, October, 1958.

MANNED SPACECRAFT SIMULATION

An Introduction to a Panel Discussion organized and moderated by

John McLeod
General Dynamics/Astronautics
San Diego, California

Computer simulation of electro/mechanical systems has become so important in our world of technical complexity that to many of us the very word "Simulation" has come to suggest first—if not exclusively—just that; the computer simulation of complex physical systems.

Would that a definition of simulation could be that concise! Or that there were a different word for each of the several major kinds of simulation.

A recent survey, in which the author requested workers in various areas of simulation to give a one-paragraph definition of the meaning of "simulation"—to them—documented a situation which has bothered many of us for some time: Simulation can mean most anything, depending on to whom you are speaking.

And Webster's Unabridged is no great help! Among the definitions given therein are: Simulation—profession meant to deceive—a counterfeit—a fraud—

Now are any of us, as practitioners of the fine art (or science!) of Simulation going to hold still for that?

Less misleading, but still illustrative of the breadth of the field, are the two main and heretofore fairly discrete branches of simulation which our panel will discuss under the combining title "Manned Spacecraft Simulation."

Today's Aerospace (née Aircraft/Automotive/Appliance) industry, has long been the leading proponent of computer simulation as a technique for the design and evaluation of their complex electro-mechanical systems.

To a lesser extent, and more recently, the areospace industry has come to rely on simulators involving computers for pilot and crew training. These are generally referred to as Operational Flight Trainers, or simply OFTs. However for some obscure reason these two methods of simulation which have so much in common have, to a large extent, gone their separate ways. But now with man's venture into space this can no longer be. Circumstances are forcing a shotgun marriage which is long overdue!

It is this union that the panel will discuss; its raison d'être, what it is, its value, its shortcomings, and our hopes for its future.

To better understand the union and its promise let us look at the partners.

Simulation for the design and evaluation of electro-mechanical systems has usually been accomplished by programming the computers to solve the mathematical equations describing the dynamics of the system under study. Systems could then be "exercised" under various synthesized conditions and parameters changed to improve or optimize the system according to some preconceived criteria. The computers used were usually general-purpose analog or digital (or combined analog-digital systems) and the optimization was done either manually (by dial-twiddling or parameter-juggling) or automatically (by programmed iterative techniques).

To minimize errors caused by discrepancies between the mathematical description of com-

ponents and their actual performance provisions were often made for inserting real "hardware" into the simulation. But man as a system component was seldom included.

Contrasted to this were the Flight Trainers in which man is of primary concern. These usually consist of special-purpose "cockpits" and special-purpose computers. The interior of the cockpit is often a faithful reproduction of that of a specific craft, complete with "live" instruments. Visual, kinesthetic, auditory, and other "cues" are generated and used to synthesize the expected environment—with varying degrees of reality. The consequences of the crew's manipulation of the controls are computed and used to control the cockpit instruments and displays, and to appropriately alter the various environmental cues.

In the past the steps from one kind of aircraft to a more advanced one were small enough to allow extrapolation from what a man could do in one to what he could be expected to accomplish successfully in the other. The danger inherent in flight testing was tolerable, the cost bearable, and failures were seldom of international import.

Not so today! Our step into space is a whopper. We cannot extrapolate. And we cannot afford flight tests: The danger to human life is too great; the dollar cost well-nigh unbearable; and failures are far too damaging to our national prestige. Our testing must be done on the ground where the man will be safe, and in a laboratory where "failures" can be carefully analyzed without the world looking over our shoulder. In short, the "flight" part of "flight test" must be simulated. We have called this kind of simulation Manned Spacecraft Simulation. And it requires a combination of the best that the other kinds of simulation have to offer.

In Manned Spacecraft Simulation the general-purpose computers and the design and evaluation aspects of physical system simulation are combined with the man-centered simulation techniques of the OFT's to accomplish what neither could do alone. Figure 1 is a photograph of one of the offsprings of this union. While Figures 2 and 3 show other kinds of spacecraft "simulators."

To discuss Manned Space Simulation, how they became involved in it, what they think about it—good *and* bad—and to take part in that most dangerous armchair sport, predicting the future, we have engineers who design and operate such simulators, psychologists and human-factors people who design and conduct experiments using them, and test-pilots who "fly" them.

I will introduce the panelists by quoting some written material which they have furnished for the purpose.

Dr. Stanley Deutsch is Chief of Systems Research and Analysis for NASA's Man-System Integration Division. As a Psychologist concerned with Systems Analyses and Human Factors he should be able to speak with authority of the requirements for manned spacecraft simulation in the U.S. Space program.

Before joining NASA Dr. Deutsch was Vice President and Chairman of the Scientific Panel, CONSAD Corporation; Head of Technical Staff for Human Factors, Missile and Space Systems Division, Douglas Aircraft Company; a Research Psychologist, Air Force Ballistic Missiles Division, Headquarters, Inglewood and Vandenberg Air Force Base; Head, Human Factors Support, Titan Program, the Martin Company; and a Research Psychologist and Project Director, U.S. Navy Electronics Laboratory, San Diego.

The greatest value of Manned Spacecraft Simulation, Dr. Deutsch says, is that "it provides an excellent system tool for the entire design, development, production, training and operation of space systems. It can be used for both ground support and manned space flight analysis and evaluation. When properly planned and implemented, simulation can be used to provide an analog of major aspects of space system development, evaluation, and operation. It provides an effective means for predicting design requirements, testing approaches, evaluating alternate configurations, and providing training equipment design and training program criteria."

Concerning the shortcomings of MSS, Dr. Deutsch states "The more the simulation becomes like the real world, the greater the cost. In many cases simulators may actually exceed the cost of the operational equipment. A better comprehension of the purposes of simulators would help reduce these costs. Simulators frequently lag the critical design and system integration phases and thus end up more as tools

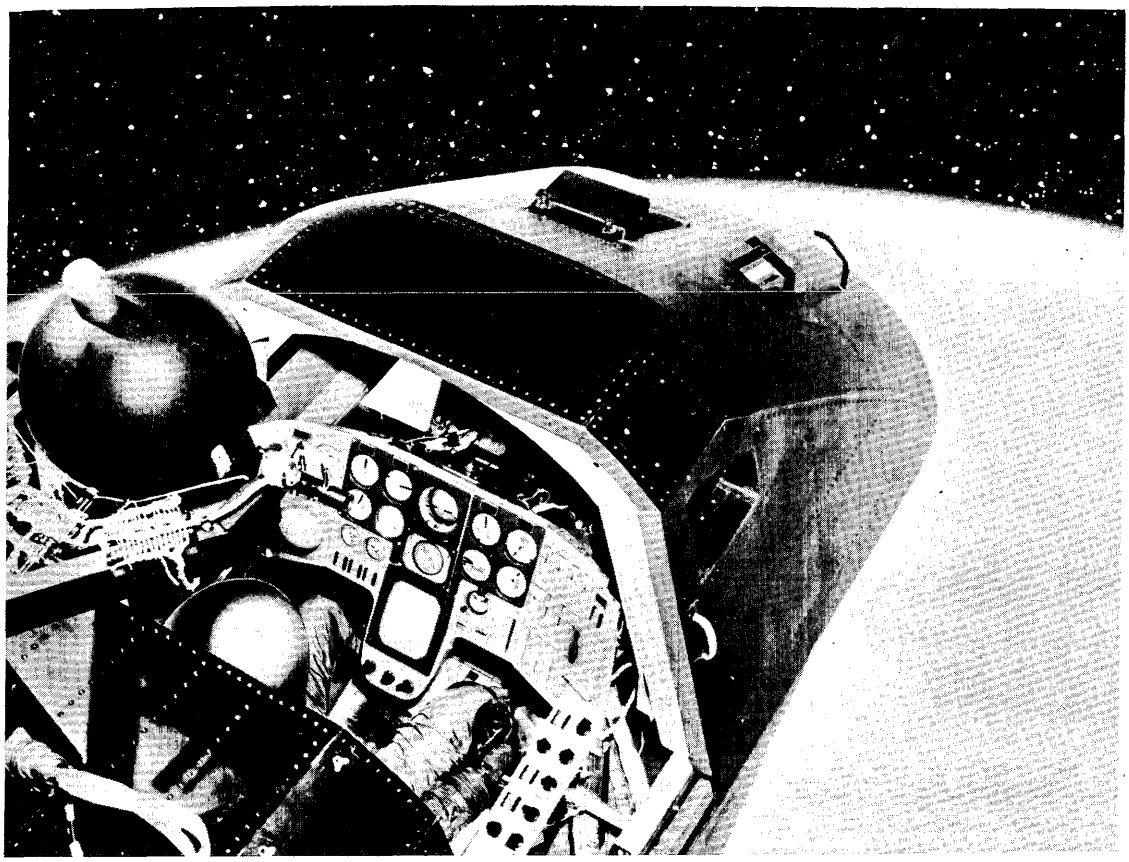


Figure 1. One example of Manned Spacecraft Simulation, showing the "gondola" and external visual cues—in this case a starfield and horizontal independently projected on the inside of a 20′ sphere. By manipulating the gondola controls the operator can duplicate any maneuver of the simulated vehicle in a real time/space condition. The impression of movement is given by the actual physical surge of the gondola along the path it would naturally take. The movement of the visual field in the opposite direction completes the psychophysiological pattern of the desired flight mode. An analog computer generates the commands to cause the proper motion of the gondola and the correct relative motion of the stars and horizon This is the Chance Vought Astronautics Division Manned Aerospace Flight Simulator in their Grand Prairie, Texas, plant.

for the evaluation of a completed design than as a concurrent adjunct to all phases of design. By the time simulation is used, many design decisions are frozen. In general, the full powers of simulation are not exploited to their greatest potential.

"However," Dr. Deutsch concludes "I feel that the many advantages of manned space simulation are commanding greater attention and consideration in all aspects of planning and research on advanced aerospace systems."

Dr. W. R. Laidlaw is a representative of industry who has had wide experience with simulation devices. He is Vice President, Advanced Systems, North American Aviation, Incorporated, Space and Information Systems Division. He has a BS in Aeronautical Engineering from the University of Toronto and an MS and ScD in Aeronautical Engineering from the Massachusetts Institute of Technology.

Dr. Laidlaw was associated with the De-Haviland Aircraft Company of Canada and the Massachusetts Institute of Technology, Aeroelastic and Structures Research Laboratory before joining North American Aviation, Incorporated in 1954. After some time with the Columbus, Ohio Division, he transferred to the Company's General Office in Los Angeles where his responsibilities included the operation of the Division's digital and analog computing facilities in which considerable effort was expended in the development of fixed and moving

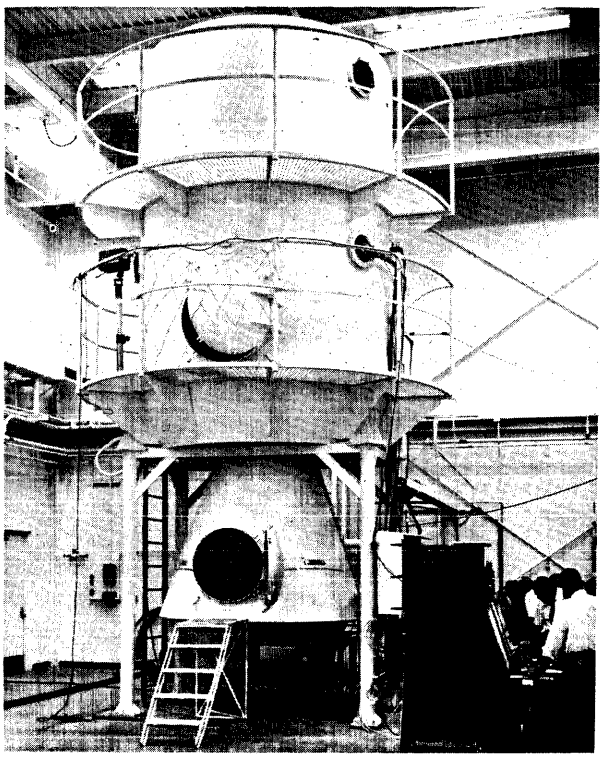


Figure 2. Another kind of Manned Spacecraft Simulator. In contrast to the kind shown in Figure 1, which is in general use to determine if and how well a man can "fly" the simulated spacecraft, this kind of simulator is used to determine if and how well a spacecraft crew can survive in space with various types of life-support equipment. Volunteer crews have spent many days in such simulators while aerospace medicine personnel use biomonitoring equipment to observe their psychological and physiological functions and human factors personnel use closed-circuit T-V to watch their every action as they perform space-type tasks, exercise, relax, and sleep. This is the 'Manned' Static Space Simulator at General Dynamics/Astronautics in San Diego.

base simulators for engineering research and development.

In 1961 he came to his present position where he is responsible for the study of future manned and unmanned space systems and launch vehicles for both scientific and military applications.

"Manned Space Simulation," says Dr. Laidlaw "has two major values. The first of these is in the vital role that it plays in the development process of complex systems by providing an environment within which the engineer can formulate the solutions to problems which would otherwise await the availability of flight test hardware. The second value of space simulation is the role which it plays in the training of flight and ground crews for the eventual operational problems which they will face.

"The combined engineering development and training features of space simulators enable significant gains from the point of view of program timing, economics, and ultimate program success.

"Perhaps the greatest shortcomings attributable to manned simulators" Dr. Laidlaw says, "is their inability to reproduce the psychological environment associated with a complex and dangerous space operation. This shortcoming can be circumvented in many instances by appropriate and careful attention to experimental design. Obviously, there remain many additional "mechanical" limitations associated with motion, optics, local environment, etc. To a high degree however, these exist because of design, hardware, or financial limitations; and, accordingly, can frequently be minimized in specific programs or applications."

The next Panelist, Mr. William B. Luton, is an engineer who has been responsible for the design, construction, and operation of one of this country's better-known Manned Spacecraft Simulators. He is Supervisor, Manned Aerospace Flight Simulator, at the Chance Vought Corporation, Dallas, Texas.

Mr. Luton is a graduate of the Univeristy of Houston, with a B.S. degree in Mechanical Engineering and of the V-5 Naval Aviation Cadet Program. He has served as a Naval Aviator and been employed as a design engineer for a scientific instrument manufacturing firm. He joined Chance Vought Corporation in 1950. He was made Project engineer in 1958 for the design and development of a Cockpit Characteristics Simulator for investigating the orbital, re-entry and hypersonic glide phases of the Dyna Soar mission. He was project engineer for a study forecasting the Air Force space crew training requirements for the next 15 years, and originated the concept for, and served as project engineer for the development of Chance Vought's Manned Aerospace Flight Simulator. He is currently supervisor of that facility.

Among Mr. Luton's writings are "Selection and Training of Space Flight Personnel" (co-author); "Manual Flight Techniques for Atmospheric Re-entries"; "The Role of Simulators for Research and Training"; "Manual Control

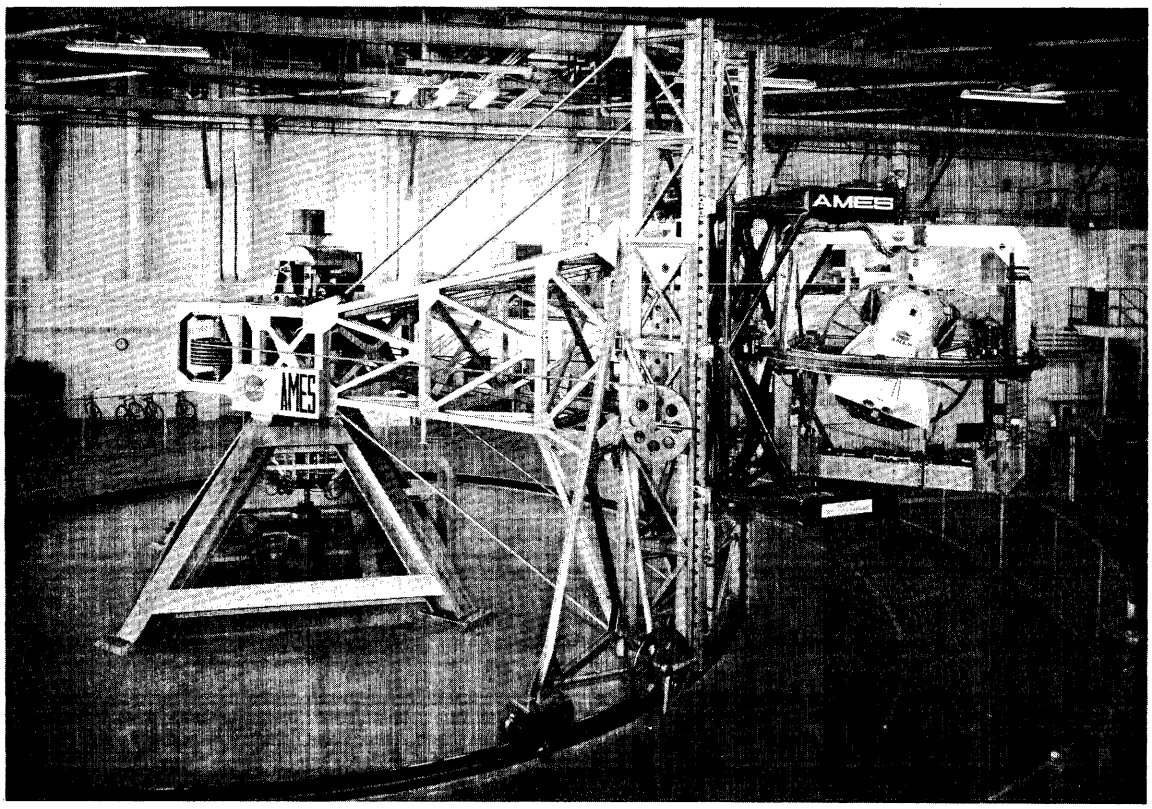


Figure 3. As indicated by the completely different configurations and functions of the simulators of Figures 1 and 2, it is often necessary to conduct "part-task" simulations. Even if it were possible to simulate everything at once the experiment—and the equipment—would be so complex as to make it impractical. This part-task simulator at NASA's Ames Research Center in California is basically a centrifuge with a man-carrying cab mounted on the arm. Although capable of creating up to 6 g. in continuous rotation, it is more often used as a five-degrees-of-freedom motion simulator. The gimballs allow three degrees of angular freedom; the vertical rails allow vertical motion; and small angular movements of the arm approximate pure lateral motion at the cab.

Underlying the basic question "Is your information valid?" is this one of part-task simulation: "Can interaction be ignored—or the results be predicted adequately?"

of Launch Vehicle Systems"; and "Space Simulators—Prelude to Manned Space Flight" (co-author).

"The greatest value of MSS," Mr. Luton writes, "lies in Human Capability Studies to provide trade-off data for use in Man-Machine Task Allocation." "Too many assumptions are being made" he says "regarding a pilot's capability and what he requires to accomplish space missions. Of almost as much value as task allocation is the use of MSS for spacecraft design, cockpit procedure and flight technique development; and crew training."

Mr. Luton considers the greatest shortcoming of MSS today to be the low fidelity of the visual and physical cues as well as stressors. This shortcoming and the absence of the "fear factor" prevents the test subject or trainee from achieving a realistic psychological and physiological behavior such as would occur in the actual case. "Quantitatively" he says, "all simulation shortcomings are perhaps no more than 10% of the total job to be done. However, a 10% deficiency in spaceflight is very costly."

Anyone for arguing Mr. Luton's last point?

The next Panelist is John M. Stroud. Mr. Stroud neither designs, uses, nor flies simulators, but he has designed many experiments with "homo sap" in the loop. And John has some very definite ideas about man—and mankind—in space.

Mr. Stroud is a member of the Operations Research Group, Command Staff, Pacific Missile Range, Point Mugu, California. His Aca-

demic training was in Economics and Psychology, but he considers himself a General Scientist.

According to his own testimony Mr. Stroud "has waged a life long battle against the pressures which are brought to bear on all men who enter the world of science to become a member of one of the species of socially approved and desired scientific specialists, a man who necessarily can act like a scientist only about some things, at some times, and hopefully will do so only as directed and when approved by proper authority." "He hopes to live long enough", he says "to see the current growing interest in the utterly imaginary creature the 'interdiciplinary specialist' (for all scientists must be specialists of some sort, at least by fiat) give rise to an acceptable profession, the General Scientist, who can attempt to act like a scientist about all things at all times, in spite of his complete incapacity to replace his specialized fellows (who can always be found vastly more competent with respect to any specific thing at any specific time)."

Mr. Stroud's interest in Manned Spacecraft Simulation grows out of his basic interest as a general scientist; the answer to the very general question of "whither mankind." "It was apparent to him, in his youth", he says, "that in the latter part of this century and the first part of the next mankind would begin to explode off the earth and would have established the first self-sustaining colony off of this earth by the middle of the twenty first century." Much of Mr. Stroud's own work in the area has been an effort "to separate the myth and social wishful thinking in these dreams of mankind from the solid scientific dreaming—which will be realized."

Mr. Stroud is of the opinion that "we still suffer from considerable lunacy in our lunar program, and from astrology in astronautics generally. The lunacy lies less in what we do than why we believe it worth doing. The astrology lies not in our developing technology, but in our almost fanatic preoccupation with major planets which are extremely difficult to colonize." "We are only very slowly and painfully learning" he says, "that Space itself is the real domain of interest."

Mr. Stroud's regularly repeated prediction is that within a thousand years there will be a thousand times the earth's present population living in the solar system, although not ten percent of this population will live *on* any planet or moon. And with slight provocation he will attempt to show how the earth could be disassembled and reassembled in a technically more useable design; one that would provide adequately for a million fold increase in population. But, he will argue, there are still more efficent methods of building in space—"the earth should be kept relatively untouched as a monument to the origins of mankind!"

"In this context manned spacecraft simulation is an essential technique. It is the only way we can afford to fail often enough to learn enough to accomplish what man envisions. Success never contributes to our wisdom. When we succeed, we merely demonstrate that we knew what we thought we knew and are none the wiser.

"All of the specific virtues of Manned Spacecraft Simulation follow from the fact that this technique can give us many failures—and they can be minutely measured and studied at low cost."

Mr. Stroud does wish to point out, however, that although "in general simulation affords the most useful failures at the least cost, it is valuable only so long as this is true—and it is not true without limit. As the art of space flight progresses, earth-bound simulation of the space environment becomes more difficult; it must be more precise and more detailed, and it must be carried out for longer periods. The cost of the simulators will rise as some large power of the dimensions of the device, and with time. And productivity will fall with time. Each test will require more time to set up, more time to execute.

"At the same time, the cost of a pound of freight to space will be declining. Probably before the end of the decade simulating the space environment will come to a halt in area after area as it becomes easier to take the test into space than to simulate space in an earth-bound test.

"Today is not too soon to demand simulation facilities in space for delivery before 1970." Mr. Stroud thinks, "In the long haul, it will matter far more which contender in the space race manages to get the first good laboratory into orbit than which first manages to get moondust on its astronauts' face plates."

Professor Paul M. Fitts is the next Panelist. Dr. Fitts is Professor of Psychology at the University of Michigan, where he also serves as Chairman of the Committee of Adaptive Systems of the Institute of Science and Technology. He is a past-president of the Society of Engineering Psychologists, currently the President of the Human Factors Society, and a member of the Board of Directors of the American Psychological Association, and of the American Institute for Research. Dr. Fitts was in the Air Force during World War II, and has served on many Government scientific advisory groups including the National Research Council Committee on Aviation Psychology, the NACA Panel on Crash Injury, the Defense Science Board, the Air Force Scientific Advisory Board, and as a consultant to the President's Science Advisory Committee. He also has been consulting psychologist on numerous assignments for industry.

Dr. Fitts observes that "simulation for use in research to determine man-machine system design characteristics poses somewhat different requirements than does simulation for crew training. However, both applications require quantitative, reliable, and valid performance measures. There is no point in simulation without the quantification of system performance.

"The issue of realism or completeness of simulation, from a psychological viewpoint (i.e., from the standpoint of developing human skills and measuring human and system performance), can only be answered on the basis of adequate theory regarding human performance characteristics. To what extent can a man time-share, or multiplex? (This relates to the question of completeness of mission simulation.) To what extent will extensive training, say up to one hundred hours, in a specific subroutine, improve overall performance that involves concurrent performance of several subroutines?

"For both research and training purposes it is necessary that simulation equipment permit the controlled variation of system parameters over a range of values. Training demands that immediate feedback to the human operator of augmented information regarding system performance be given; extensive training under such conditions is also necessary before performance data are of much value in system design.

"The greatest shortcoming of MSS today" Dr. Fitts says, "is the same as that which has plagued the simulation area for the past twenty years—lack of versatile and reliable simulation equipment in sufficient quantity to permit scientific (general) answers to be obtained to the basic issues regarding the capabilities and limitations of manual operation. It is conservatively estimated that as much as five hundred hours of quantitative data (fifty hours of training data for each of ten human (operators) are often needed in order to obtain such answers to each question that is worth answering about simulation. No amount of expert opinion from psychologists, engineers or astronauts can substitute for such data."

The next Panelist is Dr. John M. Hunt, Senior Vice President and Technical Director, Simulation and Control Group, General Precision, Incorporated, Binghamton, New York. He holds the degrees of B.S. in Engineering Physics from the University of Kansas, M.S. in Electrical Engineering from the Massachusetts Institute of Technology, and Ph.D. in Electrical Engineering from Stanford University.

Dr. Hunt served as an Electronics Officer in the United States Navy during World War II and since 1949 has been affiliated with the Link Division of General Precision, Incorporated.

At GPI where Dr. Hunt's principal activities have been in the development and application of large-scale special-purpose analog and digital computers for vehicle dynamic simulation. He holds 20 patents in this field and has a number of other patent applications pending.

Dr. Hunt became actively associated with manned spacecraft simulation as the result of the extension of the activities of Link, now actively engaged in the manufacture of large-scale manned space vehicle simulators for NASA and the Department of Defense.

Although it is difficult to single out a specific principal advantage of MSS, Dr. Hunt feels "the strongest justification for simulation might lie in the areas of over-all systems integration and training. In an inherently costly and dangerous mission necessarily involving integrated activity of many people scattered throughout the world, the opportunity for repeated and meaningful dress rehearsal before actual launch is of enormous importance. This feeling is supported by the fact that substitute procedures could be employed if necessary in most other

phases of engineering and planning underlining manned space flight if simulation techniques were not available. I find it almost inconceivable, however, Dr. Hunt says, "that the incredibly complex interrelationships of human and equipment tasks which characterize an actual mission could be effectively integrated without recourse to full-scale systems simulation.

"There is little doubt that the present state-of-the-art in generation of realistic visual and motion cues for MSS equipment seriously lags behind the state-of-the-art in simulator computation. Nevertheless, as a professional devotee of large-scale simulator computer systems and techniques, I would like to raise a small voice toward provision of highly faithful dynamic computation in MSS equipment. There is considerable justification for the belief that dynamic computation anomalies in existing simulators have contributed in part to complaints directed towards the realism of visual and motion cues. In recent years, progress in computer design is such that there is no longer significant economic justification for tolerance of inadequacies in MSS simulation computation."

The next Panelist is Mr. Wesley E. Woodson, an Engineering Psychologist (Human Engineer). He is a member of the Life Science Section at GD/A and is active in manned space vehicle and station research development—especially with reference to the human operator link and dynamic man/machine interface problems.

Mr. Woodson is author of the widely used *Human Engineering Guide for Equipment Designers,* published in 1954 by the University of California Press, and a contributing author for *Industrial Electronics Handbook,* published in 1959 by the McGraw-Hill Company.

Speaking of our topic in general, Mr. Woodson observed "one of the most serious problems we face in full integrated man-machine simulation (regardless of whether we are concerned with design, testing or training) is the fact that we are not clear in our own minds just what it is we *need* to simulate." Certainly we can list a number of items which appear to be obvious. However, when we begin to consider the inputs to the human which are dependent on his personal perceptual characteristics, it is a tough question to identify these in explicit, quantitative terms. A typical example is the oft-quoted "fear" effect. Is this something which is necessary? And is it ever possible to know if we have created it as it might actually exist in the expected operational environment?

"Another example of an area of uncertainty is the term 'realism.' Is a mirage on the desert 'real'? After all, it appears to have all of the perceived attributes of the real thing. Another factor which is poorly understood involves human imagination. A pilot 'under the hood' can imagine all sorts of things because of his instrument readings and 'seat-of-the-pants' feel. But how much of this imagination is related to the flying problem in the air as compared to the flying problem in a simulator?

"It has been demonstrated that the worst thing that can happen in simulation is to introduce something which is obviously out of character with the 'realistic environment' you are trying to create. On the other hand there is precious little evidence to date on the effect of lack of realism in the synthesized environment. In other words, is it necessary to try to accurately include all of these so-called environmental cues—with the attendant possibility of introducing artificialities?"

Asked about the Greatest Value of MSS Mr. Woodson replied: "I see two most important values: (1) we can devise and test a proposed system (including man in the loop) before we are committed to any specific design concept, (2) we can flight-test the final system without its ever "leaving the ground" (which is not only safer, but also cheaper)."

Concerning the greatest shortcomings he said: "I belive there are two primary problems today—particularly from the human engineer's point of view, i.e., (1) lack of flexibility (too long to set up new programs), and (2) too expensive to run a given exercise (in order to study a given configuration objectively, it is often necessary to run a simulator many hours)."

As the final speaker on our panel we have a test pilot, or more accurately a Research Pilot, to give us some background information about himself and some opinions on the value and shortcomings of MSS. Then perhaps in the discussion period to follow he will tell us how "flying" one of these gadgets compares with flying the real article.

Robert S. Buchanan, Major, USAF, is Chief, Research Pilot Division, Aerospace Research Pilot School, Edwards Air Force Base, California. He obtained his Ph.D. in Aeronautical and Astronautical Engineering, from the University of Michigan. He is also a graduate of the USAF Experimental Test Pilot School and Aerospace Research Pilot School. He has been a USAF Test Pilot and R & D Officer for the past ten years, and involved with Manned Spacecraft Simulation since establishment of the USAF Aerospace Research Pilot training program in 1961. He is concerned with all general areas of simulation not connected with specific vehicles (i.e., boost and ascent, rendezvous, re-entry, landing).

"The greatest value of Manned Spacecraft Simulation as I envision it" Major Buchanan states, "is in the areas of training and overall system integration. In the training phase, use should be made of special purpose trainers to simulate specific areas of operation such as orbital rendezvous and docking; also dynamic simulators such as the centrifuge to expose crew members to G-loadings at and above levels expected in flight, and under the same conditions expected in flight (i.e., eye balls in, out, down). Finally, overall system integration can be enhanced by use of the results of the part-task simulators, such as hand controller design for use under G-loads, instrument display, and cockpit or crew compartment requirements."

"The greatest shortcoming of Manned Spacecraft Simulation as I see it" he says, "is in the area of motion cues. Part-task and general purpose fixed-base simulators provide excellent training in procedures and controlling techniques; however, the motion cues as presently employed by most moving base simulators are unrealistic and of questionable value to the experienced test pilot. The centrifuge appears to be the best available method of providing experience in varying load factor and should be used in conjunction with special purpose simulation devices."

Having set the tenor of the meeting with the foregoing, the points raised are open to discussion by Panelists and Audience—and the Moderator!

REVIEWERS, PANELISTS, SESSION CHAIRMEN AND PANEL MODERATORS

AFIPS and the 1963 Spring Joint Computer Conference Committee would like to express their sincere appreciation to those listed below for their contribution toward the formulation and execution of the technical program.

REVIEWERS

C. W. Adams
J. Belzer
R. W. Bemer
R. F. Clippinger
D. Dahm
L. Drescher
F. A. Engel
A. Evans
R. A. Henle
G. B. Herzog
W. S. Hines
K. E. Iverson
V. L. Larrowe
R. M. Lee
F. N. Marzocco
E. M. McCormick
M. Minsky
T. H. Sumner
L. M. Warshawsky

PANELISTS

C. W. Adams
M. J. A. Arnow
J. Belzer
R. W. Bemer
R. S. Buchanan
H. H. Campaigne
R. F. Clippinger
S. Deutsch
D. C. Engelbart
F. A. Engel
D. C. Evans
R. R. Favreau
P. M. Fitts
M. C. Gilliland
R. T. Harnett
A. D. Hestenes
W. Hoffman
R. M. Howe
J. M. Hunt
W. R. Laidlaw
J. C. R. Licklider
A. W. Lo
W. B. Luton
E. M. McCormick
J. McCarthy
H. F. Meissinger
A. Newell
A. Opler
L. H. Peterson
O. H. Schmitt
P. J. Sehnert
N. R. Scott
O. G. Selfridge
J. N. Smith
T. D. Sterling
J. Stroud
J. W. Sweeney
L. L. Van Oosten
L. M. Warshawsky
W. H. Wattenburg
B. Waxman
M. V. Wilkes
W. E. Woodson

SESSION CHAIRMEN

H. Bromberg
D. C. Evans
E. L. Glaser
R. M. Howe
J. A. Jacquez
W. B. Kehl
J. H. McLeod
N. R. Scott
N. H. Taylor
K. M. Uglow, Jr.
L. M. Warshawsky
J. Weizenbaum

1963 SPRING JOINT COMPUTER CONFERENCE COMMITTEES

Chairman

E. CALVIN JOHNSON; Bendix Research Laboratories

Vice Chairman

DONALD E. HART; General Motors Research Laboratories

Secretary

ROBERT C. SIMS; Bendix Research Laboratories; Secretary

WALTER E. CHAPELLE; Bendix Research Laboratories; Asst. Secretary

Treasurer

DAVID V. BURCHFIELD; Touche, Ross, Bailey and Smart; Treasurer

FRANK H. TRANZOW; Touche, Ross, Bailey and Smart; Asst. Treasurer

Technical Program Committee

BRIAN W. POLLARD; Burroughs Corporation; Chairman

BERNARD A. GALLER; University of Michigan, Associate Chairman

GEORGE B. WOLFE; Burroughs Corporation; Secretary

WALTER HOFFMAN; Wayne State University

NORMAN R. SCOTT; University of Michigan

ROBERT L. SINK; Burroughs Corporation

Local Arrangements Committee

WILLIAM R. FORSYTHE; IBM Corporation; Chairman

MICHAEL A. SIEGMAN; IBM Corporation; Vice Chairman

ROBERT M. FRANKLIN; Chrysler Corporation

BRIAN E. HOLMES; IBM Corporation

ROBERT E. MANION; IBM Corporation

MARTIN P. MORTENSEN; Chrysler Corporation

Exhibits Committee

ALAN D. MEACHAM; American Data Processing, Inc.; Chairman

PAUL H. PARISSENTI; Burroughs Corporation; Vice Chairman

Printing and Mailing Committee

GWYN WILLIAMS; Michigan Bell Telephone Company; Chairman

JOHN F. PASSIFELD; Michigan Bell Telephone Company; Vice Chairman

Proceedings Committee

BARRETT HARGREAVES; General Motors Research Laboratories; Chairman

BURT E. SMITH; General Motors Research Laboratories; Vice Chairman

Registration Committee

ROBERT K. LOUDEN; IBM Corporation; Chairman

JAMES H. HUNTER; Chrysler Corporation; Vice Chairman

NORMAN H. MILLER; Ford Motor Company; Vice Chairman

Public Relations Committee

SAMUEL N. IRWIN; Data Systems Incorporated; Chairman

JOHN L. ROSE; Burroughs Corporation; Vice Chairman

JOHN L. MCKELVIE; Bendix Industrial Controls Division

Special Events Committee

GERALD LICHT; General Motors Research Laboratories; Chairman

ROBERT R. BURNS; Control Data Corporation

JERRY E. KELLEHER; General Motors Parts Division

WILLIAM T. SWEENEY; Sperry Rand Corporation

Legal Advisor

JOHN H. FILDEW. Fildew, DeGree, Fleming & Gilbride

Exhibits Management

JOHN L. WHITLOCK

Public Relations Consultant

LEWIS WINNER

Program Booklet

Campbell-Ewald Company

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

P.O. Box 1996, Santa Monica, California

Chairman, Board of Governors
DR. WILLIS H. WARE
The RAND Corporation
1700 Main Street
Santa Monica, Calif.

Executive Committee
DR. ARNOLD A. COHEN—IRE*
DR. HARRY D. HUSKEY—ACM
DR. MORRIS RUBINOFF—AIEE*

Secretary
MISS MARGARET R. FOX
National Bureau of Standards
Data Processing Systems Div.
Washington 25, D. C.

Treasurer
MR. FRANK E. HEART
Lincoln Laboratory
P. O. Box 73
Lexington 73, Mass.

Chairman-Elect
MR. J. D. MADDEN
Associate Director of Research
System Development Corporation
2500 Colorado Ave.
Santa Monica, Calif.

AIEE Directors

MR. G. L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California

MR. H. T. MARCY
IBM Corporation
112 East Post Road
White Plains, N. Y.

MR. C. A. R. KAGAN
Western Electric Co.
P. O. Box 900
Princeton, N. J.

DR. MORRIS RUBINOFF
Moore School of
Electrical Engineering
200 South 33rd St.
Philadelphia 4, Pa.

ACM Directors

MR. W. M. CARLSON
Director Technical Information
Office DDR&E
Office Secretary of Defense
Washington 25, D. C.

DR. ALAN J. PERLIS
Computation Center
Carnegie Inst. of Technology
Pittsburgh 13, Pa.

DR. H. D. HUSKEY
Acting Director
Computer Center
University of California
201 Campbell Hall
Berkeley 4, Calif.

MR. J. D. MADDEN
System Development Corp.
2500 Colorado Avenue
Santa Monica, Calif.

IRE Directors

DR. WERNER BUCHHOLZ
IBM Development Lab.
P. O. Box 390
Poughkeepsie, N. Y.

DR. ARNOLD A. COHEN
Remington Rand Univac
Univac Park
St. Paul 16, Minn.

MR. FRANK E. HEART
Lincoln Laboratory
P. O. Box 73
Lexington, Mass.

Mr. W. L. ANDERSON
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

AFIPS Representative to IFIP
MR. I. L. AUERBACH
Auerbach Corporation
1634 Arch St.
Philadelphia 3, Pa.

Simulation Councils, Inc. Observer
MR. JOHN E. SHERMAN
D-5915-102
Lockheed Missiles and Space Co.
P. O. Box 504
Sunnyvale, California

*IEEE as of 1/1/63

Society Heads

Mr. W. L. Anderson
Chairman, PGEC
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

Dr. Alan J. Perlis
President, ACM
Computation Center
Carnegie Institute of Technology
Pittsburgh 13, Pa.

Mr. C. A. R. Kagan
Chairman, Computing Devices
Committee, AIEE
Western Electric Co.
P. O. Box 900
Princeton, N. J.

Society Reps. to AFIPS

M. I. S. Coggeshall, Mgr.
Technical Operations Services
American Inst. of Elec. Engineers
245 E. 47th St.
New York 17, N. Y.

Mr. H. S. Bright
Secretary, ACM
Philco Corporation
Director of Programming
Computation Division
3900 Welsh Road
Willow Grove, Pa.

Dr. Richard M. Emberson,
Secretary
Professional Technical Groups
Institute of Electrical and
Electronics Engineers, Inc.
Box A, Lenox Hill Station
New York 21, N. Y.

Admissions

Dr. Bruce Gilchrist
IBM Corporation
590 Madison Ave.
New York 22, N. Y.

Awards

Mr. C. A. R. Kagan
Western Electric Co.
P. O. Box 900
Princeton, N. J.

Bylaws & Constitution

Mr. W. M. Carlson
Director Technical Information
Office DDR&E
Office Secretary of Defense
Washington 25, D. C.

Conference Committee

Mr. Keith W. Uncapher
The RAND Corporation
1700 Main Street
Santa Monica, Calif.

Education (Provisional)

Mr. George G. Heller
IBM Corporation
7220 Wisconsin Avenue
Bethesda 14, Maryland

Finance

Rr. Robert R. Johnson
General Electric Company
P. O. Drawer 270
Phoenix, Arizona

Planning

Dr. Morris Rubinoff
Moore School of Elec. Engineering
200 South 33rd Street
Philadelphia 4, Pa.

Publications

Mr. Joseph D. Chapline
Computer Division
Philco Corporation
3900 Welsh Road
Willow Grove, Pa.

Harry Goode Memorial Award Committee

Mr. I. L. Auerbach
Auerbach Electronics
1634 Arch St.
Philadelphia 3, Pa.

Public Relations

Mr. J. D. Madden
System Sevelopment Corporation
2500 Colorado Blvd.
Santa Monica, Calif.

Public Information Director

Mrs. Phyllis R. Huggins
P. O. Box 55
Malibu, Calif.

Social Implications of Information Processing Technology

Mr. Harry T. Larson
Aeronutronic
Division of Ford Motor Co.
P. O. Box 486
Newport Beach, Calif.

Ad Hoc Headquarters Committee

Dr. Bruce Gilchrist
IBM Corporation
590 Madison Ave.
New York 22, N. Y.

Ad Hoc Policy Committee

Dr. Jack Moshman
CEIR, Inc.
1200 Jefferson Davis Highway
Arlington 2, Va.

LIST OF EXHIBITORS

1963 SPRING JOINT COMPUTER CONFERENCE

ADDRESSOGRAPH-MULTIGRAPH CORP.
Cleveland 14, Ohio

AMERICAN DATA PROCESSING, INC.
Detroit 26, Mich.

AMERICAN TELEPHONE & TELEGRAPH CO.
New York 13, N. Y.

AMPEX CORP.
Redwood City, Calif.

ANELEX CORP.
Boston 14, Mass.

APPLIED DYNAMICS, INC.
Ann Arbor, Mich.

ASSOCIATION FOR COMPUTING MACHINERY
New York 21, N. Y.

AULT MAGNETICS, INC.
Minneapolis 29, Minn.

BELL & HOWELL CO., MICRODATA DIV.
Chicago 45, Ill.

BENSON-LEHNER CORP.
Van Nuys, Calif.

BRYANT COMPUTER PRODUCTS
Walled Lake, Mich.

BURROUGHS CORP.
Detroit 32, Mich.

CALIFORNIA COMPUTER PRODUCTS, INC.
Anaheim, Calif.

CAMBRIDGE COMMUNICATIONS CORP.
Cambridge 42, Mass.

COLLINS RADIO CO.
Dallas, Texas

COMCOR, INC.
Denver 23, Colo.

COMPUTER CONTROL CO., INC.
Framingham, Mass.

COMPUTER DESIGN
Boston 16, Mass.

COMPUTER SYSTEMS, INC.
Fort Washington, Pa.

COMPUTERS & AUTOMATION
Newtonville 60, Mass.

CONSOLIDATED ELECTRODYNAMICS CORP.
Pasadena, Calif.

CONTROL DATA CORP.
Minneapolis 20, Minn.

DI/AN CONTROLS, INC.
Boston 25, Mass.

DATAMATION MAGAZINE
New York 17, N. Y.

DATAMEC CORP.
Mountain View, Calif.

DATA PRODUCTS CORP.
Culver City, Calif.

DATA SYSTEMS, INC.
Grosse Pointe Woods 36, Mich.

DATATROL CORP.
Silver Spring, Md.

DIGITAL EQUIPMENT CORP.
Maynard, Mass.

DIGITRONICS CORP.
Albertson, L. I., N. Y.

DYMEC DIV., HEWLETT-PACKARD CO.
Palo Alto, Calif.

EDP WEEKLY INDUSTRY REPORTS, INC.
Washington, D. C.

ELECTRONIC ASSOCIATES INC.
Long Branch, N. J.

ENGINEERED ELECTRONICS CO.
Santa Ana, Calif.

FABRI-TEK, INC.
Amery, Wisc.

FERRANTI ELECTRIC, INC.
Plainview, L. I., N. Y.

FLOATING FLOORS, INC.
New York 17, N. Y.

FORD MOTOR CO.
Aeronutronic Div.
Newport Beach, Calif.

GPS INSTRUMENT CO., INC.
Newton 64, Mass.

GENERAL DYNAMICS/ELECTRONICS
San Diego 12, Calif.

GENERAL ELECTRIC COMPUTER DEPT.
Phoenix, Ariz.

THE GERBER SCIENTIFIC INSTRUMENT CO.
Hartford, Conn.

H-W ELECTRONICS, INC.
Natick, Mass.

HOLLEY COMPUTER PRODUCTS CO.
Warren, Mich.

INDIANA GENERAL CORP.
Keasbey, N. J.

INFORMATION SYSTEMS GROUP GENERAL PRECISION, INC.
Glendale, Calif.

THE INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS
New York, N. Y.

INTERCONTINENTAL INSTRUMENTS, INC.
Farmingdale, L. I., N. Y.

INTERNATIONAL BUSINESS MACHINES CORP.
New York 22, N. Y.

JOHN WILEY & SONS, INC.
New York 16, N. Y.

LFE ELECTRONICS, DIV. OF LABORATORY FOR ELECTRONICS, INC.
Boston 15, Mass.

LISKEY ALUMINUM, INC.
Glen Burnie, Md.

LITTON SYSTEMS, INC.
Beverly Hills, Calif.

LOCKHEED ELECTRONICS CO. AVIONICS & INDUSTRIAL PRODUCTS DIV.
Los Angeles 22, Calif.

McGRAW-HILL BOOK CO., INC.
New York 36, N. Y.

MEMOREX CORP.
Santa Clara, Calif.

MICHIGAN STATE UNIVERSITY
East Lansing, Mich.

THE NATIONAL CASH REGISTER CO.
Dayton 9, Ohio

NAVIGATION COMPUTER CORP.
Norristown, Pa.

OMNITRONICS, INC.
Sub. of Borg-Warner Corp.
Philadelphia 23, Pa.

PACKARD BELL COMPUTER CORP.
Los Angeles 25, Calif.

PHILCO COMPUTER DIV.
Willow Grove, Pa.

PHOTOCIRCUITS CORP.
Glen Cove, N. Y.

POTTER INSTRUMENT CO., INC.
Plainview, L. I., N. Y.

PRENTICE-HALL, INC.
Englewood Cliffs, N. J.

RADIO CORP. OF AMERICA SEMICONDUCTOR & MATERIALS DIV.
Somerville, N. J.

RAYTHEON CO.
Waltham 54, Mass.

RECORDAK CORP.
New York 3, N. Y.

RHEEM ELECTRONICS CORP.
Los Angeles 45, Calif.

RIDGEWAY ASSOCIATES, INC.
Chicago 34, Ill.

ROYAL MCBEE CORP.
New York, N. Y.

THE SERVICE BUREAU CORP.
New York, N. Y.

SIMULATION COUNCILS, INC.

SOROBAN ENGINEERING, INC.
Melbourne, Fla.

SPARTAN BOOKS, INC.
Baltimore 1, Md.

SPRAGUE ELECTRIC CO.
North Adams, Mass.

TALLY REGISTER CORP.
Seattle 9, Wash.

TELETYPE CORP.
Skokie, Ill.

US INDUSTRIES, INC. EDUCATIONAL SCIENCE DIV.
New York, N. Y.

UNIVERSITY OF MICHIGAN
Ann Arbor, Mich.

WARNER ELECTRIC BRAKE & CLUTCH CO.
Beloit, Wisc.

WAYNE STATE UNIVERSITY
Detroit, Mich.

WESTINGHOUSE ELECTRIC CORP.
Pittsburgh 30, Pa.

AUTHOR INDEX