Home Copy

see 1134

FP Post

U. S. NAVAL ORDNANCE LABORATORY
White Oak, Silver Spring 19, Maryland

NAVAL ORDNANCE LABORATORY MEMORANDUM 10337

29 March 1950

From: H. B. Curry
To: NOL Files
Via: Chief, Applied Mathematics Division
Aeroballistics Research Department

Subj: A Program Composition Technique as Applied to Inverse Interpolation. (Project NR 044-049*)

Abstract: The research into program composition, begun in reference (a), is here continued. The technique of program composition is applied to the inverse interpolation problem for two variables in the form in which it is useful for the calculation of a firing table from trajectory data. This problem is analyzed and then exhibited as a composite of certain component programs called divisions. The synthesis of the main program from those for the divisions is discussed in outline, and that of certain of the divisions from their ultimate constituents is considered in detail. Standard methods of performing arithmetic operations, controlling iterations, testing betweenness relations, and reading from tables are proposed. There results a technique of program construction which promises some improvement over existing procedures; if developed further it is probable that the efficiency of a computing establishment can be increased thereby. Certain considerations affecting details of the design of the machine also arise in these studies.

Foreword: The data and conclusions presented here are the opinions of the author and do not necessarily represent the final judgment of the Laboratory.

Ref:
(a) Curry, H. B. On the Composition of Programs for Automatic Computing. NOLM 9805 (1949).
(b) Curry, H. B. and Wyatt, Willa. A Study of Inverse Interpolation on the ENIAC. Aberdeen Proving Ground, Ballistic Research Laboratories, Report No. 615, (1946).
(c) Goldstine, Hermann H. and von Neumann, John. Planning and Coding of Problems for an Electronic Computing Instrument. Part II, I, Institute for Advanced Study (1947).

*Sponsored by the Office of Naval Research

HASKELL B. CURRY
[illegible]

(d) Ibid. Part II, III (1948).
(e) Lotkin, Max. Inversion on the ENIAC Using Osculatory Interpolation. Aberdeen Proving Ground, Ballistic Research Laboratories, Report No. 632 (1947).
(f) Quade, W. Ein neues Verfahren der schrittweisen Naherungen zur Losung von $Y' = f(x\ y)$. Mathematische Zeitschrift, 48, 324-368 (1942).

Encl: (A) Appendix
(B) Charts A through C

Introduction

1. The present memorandum continues an investigation into program composition for automatic computing machines which was begun in a previous one, reference (a). The entire investigation, including both the previous memorandum and this one, was motivated by, and is based on, a study of the problem of inverse interpolation which was begun in a report, reference (b), issued at Aberdeen Proving Ground. That problem was selected for special study, in spite of its relatively simple character, because it contains a wide variety of kinds of program composition; such a wide variety, indeed, that it is not unlikely that most of the kinds of composition which one is apt to meet in practice will be exemplified there. Whether or not this is so is not relevant; for if other kinds of composition should be found necessary, they can be added to those considered in these memoranda afterwards. It is certain that the problem contains many kinds of common occurrence; and that a considerable number of the more complex computations, such as the step-by-step solution of differential equations, can be programmed by the aid of the principles developed here.

2. The dependence of this investigation on the inverse interpolation problem will become more obvious in this memorandum than it was in the previous one. This is because of the effect of some accidental factors in the management of this investigation. The first step in the study, as planned, was to analyze the inverse interpolation problem into its main constituent parts, and then to study the kinds of composition necessary to reconstruct the program from these main parts. The first memorandum considered, besides certain generalities of the preliminary nature, the kinds of composition arising in this first stage. It was not possible to include the corresponding synthesis, here called the main synthesis, in the first memorandum. Practically all of the research reported on there was done in June 1948; and the memorandum, which was first submitted July 26, 1948, contained as much as it was possible to put in it of what had been done up to that time. Thus the main synthesis, although conceived as a part of the first memorandum, had to be delayed so as to form Chapter I of the present one. It will be seen that the main synthesis is, essentially, a special case of the kind of composition considered in reference (a), Section IV E.

3. Beyond this point there is the consideration of how these major parts, here called divisions, may be compounded from simpler parts. This analysis, and the corresponding synthesis, occupies the later chapters of this report.

4. This analysis can, in principle at least, be carried clear down until the ultimate constituents are the simplest possible programs. These programs, which are here called basic programs, each consist of a simple order plus the necessary outputs and data. Of course, it is a platitude that the practical man would not be interested in composition techniques for programs of such simplicity, but it is a common experience in mathematics that one can deepen one's insight into the most profound and abstract theories by considering trivially simple examples. Accordingly, the objective here will be to build up the inverse interpolation program from the basic programs themselves. This objective will be approached as closely as time permits. Whether or not an improvement in methods of forming simple programs from scratch will be attained in this way will be left undecided.

5. It is necessary to specify these basic programs rather more in detail than was done in the first memorandum. The discussion of them will occupy Chapter II. That will be the place to introduce other assumptions in regard to the machine, and to make suggestions in regard to additional sorts of basic programs.

6. The synthesis of further programs from the basic programs will be considered in Chapters III, IV, and V. The first of these will deal with "arithmetic programs", i.e., programs which do not involve any discriminations or partial substitutions. In this case a more or less complete theory for the construction of an arbitrary such program will be given. This program will not always be the shortest one possible to attain the required result; but, at least, it will be automatic as soon as certain decisions are made. In Chapter IV discrimination programs will be considered. Here the emphasis will be upon the kinds of discriminations needed in the inverse interpolation problem. A composite program will be constructed in this chapter which will give, so to speak, the complete logical structure of the inverse interpolation problem. Chapter V will be devoted to secondary programs. These involve what Goldstine and von Neumann (Reference (c)) call "variable and connections". It will be found that another form of program composition will be necessary in this chapter.

7. Since the first memorandum was written, the third part (Reference (d)) of the report of von Neumann and Goldstine has appeared. This report involves rather less overlapping with the present investigation than had, at first, been anticipated. Those authors have a point of view sufficiently different from the present one so

that it is not immediately obvious just how the two investigations should be put together. Accordingly, it seems advisable to continue the independence of this investigation, at least for the time being. One aspect of the problem, however, is that it is no longer necessary to consider, viz., the question of how a program composition can be effected by means of the machine itself. In Reference (d) there is given a preparatory program for carrying out on the main machine a rather complicated kind of program composition. But one comment seems to be in order in regard to this arrangement. The scheme allows certain data to be inserted directly into the machine by means of a typewriter-like device. Such an arrangement is very desirable for trouble-shooting and computations of a tentative sort, but for final computations of major importance it would seem preferable to proceed entirely from a program or programs recorded in permanent form, not subject to erasure, such that the computation can be repeated automatically, or at least in a uniquely determined manner, on the basis of the record.* It is supposed here that a file of such programs will be built up, and the objective is to form complex programs of that sort from simpler ones. Of course the procedure would be useable under other circumstances.

8. Two incidental points in regard to this project need to be emphasized in order to avoid misunderstanding.

9. In the first place the present memorandum is not intended as a contribution to the technique of inverse interpolation. That problem is considered here because it is interesting from the standpoint of program composition. The method of carrying out the inverse interpolation is also chosen from that same point of view. Although a number of refinements are introduced to make the scheme adequate for inverse interpolation, the question of whether this program is the most economical for that purpose is not even considered.

10. In the second place the idealizations and assumptions made in Reference (a) are maintained here. However, in the programs constructed due regard is paid to the consecutivity of orders, i.e., the restriction that the exit of an order is always the next following order, except that unconditional control shifts and stop orders have no exits, is adopted. It seems hardly necessary to discuss in detail the modifications in the theory of Reference (a) due to this circumstance. The discussions in Reference (a), Chapter IV, were planned with this eventuality in mind, and in every case it can be seen, by putting in the exit numbers explicitly, that the theory of Reference (a) applies.

---

*It is said that during the war an error in one of the firing tables was caused by using the wrong lead screw in the differential analyzer. Such an error would have been impossible if the calculation had been completely programmed.

11. The investigation reported on in this memorandum is unfinished. The objective was to create a programming technique based on a systematic logical theory. Such a theory has the same advantages here that it has in other fields of human endeavor. Toward that objective only a beginning has been made. Further study of the matter will doubtless lead to essential improvements. Moreover, considerations affecting the design of the machine are likely to arise, so that it is advantageous that such studies be prosecuted before the designs are completely frozen. Efficiency in the management of an eventual computing enterprise can no doubt be furthered by such a study as this, but if it is to have that effect it must be pursued while the plans are still in the formative stage, and it must be carried beyond the stage of preliminary analysis to the point where it can be tried on practical problems.

CHAPTER I

The Inverse Interpolation Problem and its Main Synthesis

A. Formulation of the Problem

12. The problem of inverse interpolation may be formulated as follows. There is given a table of the functions

$$y_\mu = f_\mu (x, \phi) \tag{1}$$

for the arguments

$$x = x_0 + \kappa \Delta x, \qquad \phi = \phi_0 + \lambda \Delta \phi, \tag{2}$$

where κ and λ range over certain segments of the integers. It is required to tabulate certain functions

$$z_\mu = g_\mu (a, \phi) \tag{3}$$

for the values

$$a = a_0 + \rho \Delta a, \qquad \phi = \phi_0 + \lambda \Delta \phi, \tag{4}$$

in which ρ and λ range over certain segments of the integers, the range for λ being the same as before. These functions g_μ are to be calculated so as to satisfy the identities

$$f_0[g_0(a,\phi),\phi] = a, \tag{5}$$

$$g_\mu(a,\phi) = f_\mu[g_0(a,\phi),\phi]. \tag{6}$$

13. This problem was originally considered in Reference (b) with reference to the calculation of firing tables from trajectory data. Here the data (1) represent the coordinates of the shell, and possibly the components of its velocity and acceleration, as functions of the time of flight t and angle of departure. ϕ. Now it is to be noted that the quantities a and y_0 have the same physical meaning, as do also the quantities t and z_0, y_μ and g_μ for $\mu > 0$. Thus if y_0 is one of the coordinates of the projectile, say its abscissa, then the final equations (3) will give the time of flight, as well as the other y's, as functions of y_0 and ϕ. Two successive applications of this process will give a firing table, i.e., a table listing t and ϕ possibly along with certain of the y's as functions of the coordinates of the target.

14. The problem will be considered here, however, entirely apart from the above physical application. Likewise only a single inverse interpolation will be considered. It is evident that by program composition two or more of these inverse interpolation processes can be combined, and so on ad infinitum; the line will be drawn at a single complete process. However, the formulation will be carried through without specification as to the number of values of κ, λ, μ, or ρ; so that certain numerical constants k, l, m, r must be given in the quantity program. Furthermore we shall adhere to the restriction, motivated by the above interpretation, that the inequalities

$$f'(x,\phi) \geqq 0 \tag{7}$$

$$f''(x,\phi) < 0, \tag{8}$$

where f' and f'' are the first and second derivatives of f with respect to t for fixed ϕ, hold throughout the range considered.

B. Analysis of the Computation

15. We now consider the carrying out of such an inverse interpolation circulation, without regard to the nature of the instruments used.

16. It is evident that the determination of the z_μ for a given fixed value of ϕ requires only the use of the equations (1) for the same value of ϕ. Thus we can carry through the computation with ϕ determined by (2) with successive fixed values of λ. Each such calculation for a single λ will be called a group.

17. Within any one group we have essentially an inverse interpolation problem for functions of one variable. We solve the equations (5) and (6) for the successive values of a given by (4). Each such calculation for a fixed a will be called a round.

18. Within any round the parts for fixed μ are distinct. Each such part will be called a phase. The phase for $\mu = 0$ will consist of the solution of the equation (5) for the given values a and ϕ; this is inverse interpolation. The phases $\mu = 0$ are calculations from (6); these are all direct interpolations.

19. Before considering the further analysis of a phase we must make some general statements about interpolation.

20. Let $f(t)$ be any one of the functions (1) considered as a function of t only. Then we seek a polynomial $P_\kappa(t)$ which forms a satisfactory approximation to $f(t)$ for

$$t_0 + \kappa \Delta t \leqq t < t_0 + (\kappa + 1) \Delta t . \tag{9}$$

It is convenient to take a variable u such that

$$t = t_0 + (\kappa + u) \Delta t \tag{10}$$

i.e.,

$$u = \frac{t - t_0 - \kappa \Delta t}{\Delta t}$$

Then, supposing a fixed value of κ, we set

$$F(\mu) = f[t_0 + (K + \mu)\Delta t]. \tag{11}$$

We shall then call the approximating polynomial P(u) and suppose that it forms a satisfactory approximation to F(u) for

$$0 \leqq \mu < 1. \tag{12}$$

We shall further suppose that P(u) is determined linearly by the data in the range

$$-p \leqq u \leqq q. \tag{13}$$

Ordinarily P(u) will be determined solely by the values of F(u) for integral u in the range (13); this will be called the normal situation. But it is conceivable that P(u) may be determined by the values of all the functions (1) in the range (9) corresponding to (13); any such case will be called a generalized situation. The osculatory interpolation considered by Lotkin in Reference (e) is an instance of a generalized situation.

21. In the future the term calculation will be reserved for the computation of such a P(u) for a given value of u. Such a calculation will involve two parts, viz.: (1) the computation of coefficients, etc., which do not depend on the value of u, and (2) the final computation. Those parts will be called the constant and variable parts, respectively. In case the calculation has to be iterated for a new value of u, only the variable part needs to be repeated.

22. We return now to the analysis of the computation of a phase.

23. Evidently a phase for $\mu=0$ will consist of a single calculation. In the normal situation we shall suppose that the P(u) is constructed in the same way in all the phases, so that in a new phase we merely repeat the same calculation program with different data. In a generalized situation there may be a different calculation program for each phase. One possible case is that where the phases are grouped in sets, with the same calculation program in each set. This is of some interest in case functions (1) are x, $\dot{x}$, $\dot{x}^\circ$, y, $\dot{y}$, $\dot{y}^\circ$, etc.; then

x, $\dot{x}$, $\dot{x}^{\circ}$ all use the same data, while the pairs x,y and $\dot{x},\dot{y}$, etc., all use the same calculation program. This would require the consideration of stages intermediate between a phase and a round. It is not profitable to consider how to program all such possibilities.

24. In the phase $\mu=0$ the first step is to determine a value of κ such that there will be a root of the equation (5) in the range (9). If we set, ignoring the argument ϕ,

$$x_2 = f_0(x + (\kappa + 2)\Delta t), \tag{14}$$

the value of κ will be correct if

$$x_0 \leqq a < x_1. \tag{15}$$

Under the assumption (7) there will then be at most one root of (5) for the entire range of our computations; hence, if (15) is satisfied, there will be a unique root in the range (9). The value of κ so determined will be called the <u>state</u>, and the part of the computation concerned with determining it will be called the <u>state determination</u>.

25. Once the state has been determined there are at least three procedures open for the further computation in phase $\mu=0$, viz.: (a) We may use a Lagrange interpolation formula. If we use this formula without restriction to equidistant arguments, it is well known that inverse interpolation is no different in principle from direct. In this case the further computation of phase $\mu=0$ consists of a single calculation with the roles of dependent and independent variable reversed. (b) We may use an iteration process of the form

$$u_{\nu+1} = u_\nu - c_\nu [P(u_\nu) - a] \tag{16}$$

where P(u) is the polynomial approximation to $F(u) = f_0(x_0 + (\kappa+u)\Delta t)$, and the c_ν are constants suitably determined. This is the process considered in Reference (b), Section 3. (c) We may use an iteration which proceeds digit by digit, as in Horner's method or ordinary long division. This is the process considered in Reference (b), Section 9.

26. In the present memorandum only the possibility (b) is considered. The sole reason for this is that it seems most interesting from the standpoint of program composition. Then the computation for

phase $\mu=0$ will consist of an iterated series of calculations. The iterations, which involve the variable part only, are under the control of a program which we shall call inversion control.

C. General Assumptions Concerning the Program

27. In order to realize such a schedule, we shall have to have a more specific plan than that sketched above. Accordingly, we shall make the following special assumptions:

(a) We consider the normal situation only.

(b) We shall suppose that λ ranges from 0 to 1, ρ from 0 to r, μ from 0 to m and κ from 0 to k. The program will work for arbitrary values of these letters, except that we must have $k > p + q$.

(c) Our program will be that of the "basic scheme" of Reference (b). But we shall include safeguards which will confine κ to the range

$$p \leqq \kappa \leqq k - q, \tag{17}$$

and also a safeguard against riding over a maximum of $f_0(t)$ (i.e., for stopping a group when (9) fails to hold). But we shall not include any auxiliary test.*

(d) As required in the basic program we shall include in the inversion control various safeguards. We shall insist on monotone convergence, shall require that (13) hold and that the number of iterations shall not exceed n (which is supposedly given in the data of the problem). Likewise we shall require that all c_ν be the same, so that we write $c_\nu = c$. This c is also given among the constants of the problem.

(e) In view of the safeguards required in (c) and (d), it may be necessary to terminate the computation, or some part of it, due to various abnormal conditions. Some of these abnormalities may be due to errors in the formulation of the problem or may indicate malfunctioning of the machine;

*An auxiliary test of some kind is likely to be necessary for any practical realization of the program. The assumption made here, that k is the same for all λ, is unrealistic, and an auxiliary test would be one way of taking care of this departure from reality.

in these cases it is appropriate to stop the machine (by bringing the control to a special output). The output and the configuration of the machine will then show the reason for the stoppage. But others of these abnormalities will occur in ordinary operation due to explainable causes. In such cases it seems better to have the machine make a note of the abnormality and proceed to the next group or round. This can be done by providing a location in the memory for receiving such a signal. This location will be called the group or round error register. The signals themselves can be various digits, and can be generated in the machine or given in the data of the problem. If printing of results is to be provided for, the error register should be printed at the conclusion of a group or round to show whether the component was normal or abnormal.

D. General Characteristics of the Program

28. The program will be divided into sections designed by capital letters according to the following scheme:

A. Orders, other than storage orders.

B. Location numbers, i.e., storage orders.

C. Given quantities, other than values of the functions.

D. Quantities calculated in the course of the computation.

E. Working locations, i.e, locations reserved for temporary storage of the x_i (defined by 14).

F. Storage of the functional values:

$$F_{\mu\kappa\lambda} = f_\mu (t_0 + \kappa \Delta t, \phi_0 + \lambda \Delta \phi) \qquad (18)$$

G. Storage of the answers:

$$G_{\mu\rho\lambda} = g_\mu (a_0 + \rho \Delta t, \phi_0 + \lambda \Delta \phi). \qquad (19)$$

H. Other answers.

29. The locations A-E in the above list will be supposed to be in the active memory of the machine; the location F, G, H may or may not be. We shall study the program as if these locations were all in the active memory; nevertheless, we shall regard the transfer of a quantity to G or H as a printing of that quantity. Transfers will be made to these locations, but never from them.

30. Evidently the locations A-B will constitute the order program, the location C-H the quantity program. The locations C and F will contain the data of the problem, G and H the answers. The locations C and F will be kept unchanged throughout the computation.

31. In the following we shall let e, f, g, h denote the location numbers of the initial words of E, F, G, H, respectively. Then we have:

$$\begin{aligned} f &= e+p+q+1 \\ g &= f+(m+1)(k+1)(\ell+1) \\ h &= g+(m+1)(r+1)(\ell+1) \end{aligned} \tag{20}$$

32. Further the locations of x_i, $F_{\mu\kappa\lambda}$, $G_{\mu\rho\lambda}$ will be, respectively,

$$\begin{aligned} & e + \kappa + i \\ & f + \mu + \kappa(m+1) + \lambda(m+1)(k+1) \\ & \qquad = f + \mu + (m+1)[\kappa + \lambda(k+1)] \\ & g + \mu + \rho(m+1) + \lambda(m+1)(r+1) \\ & \qquad = g + \mu + (m+1)[\kappa + \lambda(r+1)] . \end{aligned} \tag{21}$$

33. As for B-D we shall not specify the exact location of their contents at this point. Instead we shall require simply that B contain e, f, g, and h* and that C, D contain the quantities listed in Table I. Here v is the location for the result of a calculation and $\$ = p+q$; the other unexplained symbols have been defined in the preceding discussion. These locations will be referred to by the same symbols as those used for their contents.

Table I

	Integers	Physical Quantities	Misc.
C	k, l, m, n, p, q, r	$\phi_0, t_0, a_0, \Delta\phi, \Delta t, \Delta a$ c, ϵ	Error signals
D	$\kappa, \lambda, \mu, \nu$ ρ, i, j, s	ϕ, t, a, u, ω	Error registers

*Even this is not necessary if we use the orders Gh as in the schedules of Chapter V. In that case there will be no locations B in the program

E. Major Divisions and Their Specification

34. According to the principles stated in the introduction, the first step in planning the program is to analyze the computation into certain main parts, called here divisions, such that the program can be synthesized from them. These main parts must be such that they, or at any rate some of them, are independent computations in their own right, or are modifications of such computations.

35. If we examine the analysis made in Section B above, we find that there are two processes, viz., calculation and inversion control, which presumably will be of interest in other connections. Likewise state determination, inasmuch as it is an elaboration of a bracketing procedure, is likely to be a modification of a process having application elsewhere. These three, then, we adopt as major divisions, and we suppose, for the moment, that we have programs for them.

36. If we go beyond this, we see that there are four ranks of parts or stages into which the computation may be divided, viz., phases, rounds, groups, and the complete computation. In any stage of each of these ranks it will be necessary to do the following four things:

A. Start the stage, i.e., make the computation preliminary to the first stage of next lower rank, set the appropriate index ($\lambda, \rho, \mu, \kappa$) to its initial value, clear the error register, etc.

B. Control the iteration of the stages of next lower rank.

C. Terminate the stage, which includes printing of results, including the error register.

D. Register an error signal or take other appropriate action in case of abnormal termination of the stage.

These actions are very similar in the different ranks. In the highest rank, the main computation, there will be no need of Parts C and D; while in the lowest rank, that of a phase, there will be no substages to be controlled. But even in the case of a phase there will be calculations to be controlled, with or without the intervention of Inversion Control. Thus for each of the four ranks there will be a program for effecting these actions, and the programs will be similar, especially for the three higher ranks in Section B. Such a program we shall call a control program.

37. The preceding considerations suggest that our major divisions should consist of four ranks of control programs together with the three listed in par. 35, giving seven main divisions as follows:

I. Main Control,
II. Group Control,
III. Round Control,
IV. Phase Control,

V. State Determination,
VI. Calculation, and
VII. Inversion Control.

38. In order to synthesize the complete program from these main divisions, it is necessary to consider the inputs and outputs of these divisions. This will occupy the next few paragraphs. The results are summarized in Table II.

39. The control programs II, III will each have three inputs and two outputs. The three inputs will be as follows:

1. To start the stage (from next higher stage).
2. To take appropriate action at termination of a lower stage.
3. To take appropriate action if the stage is abnormally terminated.

The outputs will be:

1. To start the next lower stage.
2. To terminate the stage.

40. In the cases I and IV this situation will be modified. In I the only difference will be that there is no input. In IV there will be three analogous inputs. The first input, to start the phase, may come in from III with a tentative value of κ. In phase $\mu = 0$ this will have to be checked in V. If V changes the value of κ, the phase setup (IV A) will have to be done all over again, hence the first input to IV may receive from V also. After the state has been determined for $\mu = 0$, and in any case for $\mu > 0$, a calculation will be ordered. The output to this calculation must come back to IV on a second input; this will be a signal to terminate the phase if $\mu > 0$, but to turn over the control of the calculations to VII if $\mu = 0$. After VII has finished there will be a third input to IV; and this third input will take the place of an abnormal phase termination. For IV there will be, therefore, four outputs, viz.

1. To order a state determination if $\mu = 0$.
2. To order a calculation if $\mu > 0$.
3. To bring in VII.
4. To signal that the phase is terminated.

These four outputs will go to the inputs listed in Table II.

41. In V there will be only one input, viz., from IV. There will be a variety of outputs according to a complicated series of discriminations. If (15) is satisfied, V will of course order a calculation. If $a < \kappa$, the procedure in Reference (b) was to terminate the round. Here this will be modified so as to allow an irregular calculation if

$$\varkappa_{-\varphi} \leqq a < \varkappa_0 . \tag{22}$$

In that case V will cause a modification in VII so as to relax the restriction to monotone convergence before giving the signal to proceed. If $\varkappa_1 \leqq a$, $\underline{V}$ will also order an irregular calculation if $\varkappa_2 \leqq \varkappa_1$; this prevents going over a maximum into a region where (7) no longer holds (cf. Reference (b), section 8). If $\varkappa_1 < \varkappa_2$, $\underline{V}$ must test to see whether κ can be increased. If it can, V must order such an increase, restore u to zero, and go back to IV. If it cannot, then it will also order an irregular calculation, leaving the possible necessity for terminating the group to be taken care of in VII. Thus there will be three types of irregular calculations; these can be distinguished without the necessity of an error signal simply by requiring that u and κ be printed (in H) with each round. (The irregular calculations will have a u outside the range 13). There will then be three outputs to V, viz.

1. To order a calculation.
2. To order a new phase setup (after increasing κ).
3. To order an abnormal round termination (for $a < \varkappa_{-\varphi}$).

42. The division VI will have two inputs, viz., to order a complete calculation and to order the variable part only. The former will receive its signal from IV or V, the latter from VII. There will be only one output, viz., to IV, since it is necessary to report to Phase Control for discrimination as to whether $\mu > 0$ or $\mu = 0$.

43. The division VII is only called in on orders from IV. Hence it has only one input. There are a variety of outputs, as follows

1. To order a new iteration. This occurs if $\varphi(\omega) > \epsilon$, $\nu < n$, and (13) holds.
2. To signal termination of the iteration, if $|\varphi(\omega)| < \epsilon_1$.
3. To signal abnormal termination of the group if $\omega > \varphi$.
4. Error signal, if $\varphi(\omega) < -\epsilon$.
5. Error signal, if $\nu > n$.
6. Error signal for $\omega < -\varphi$.

44. These specifications are summarized in Table II. Here the inputs to the different divisions are indicated by subscripts, the outputs by O with subscripts. The outputs O_1, O_2, O_3, O_4 without indication of division, are the outputs of the composite computation. Of these O_1 is the output for normal completion of the program; the outputs O_2, O_3 and O_4 are error signals.

Table II

Inputs and Outputs

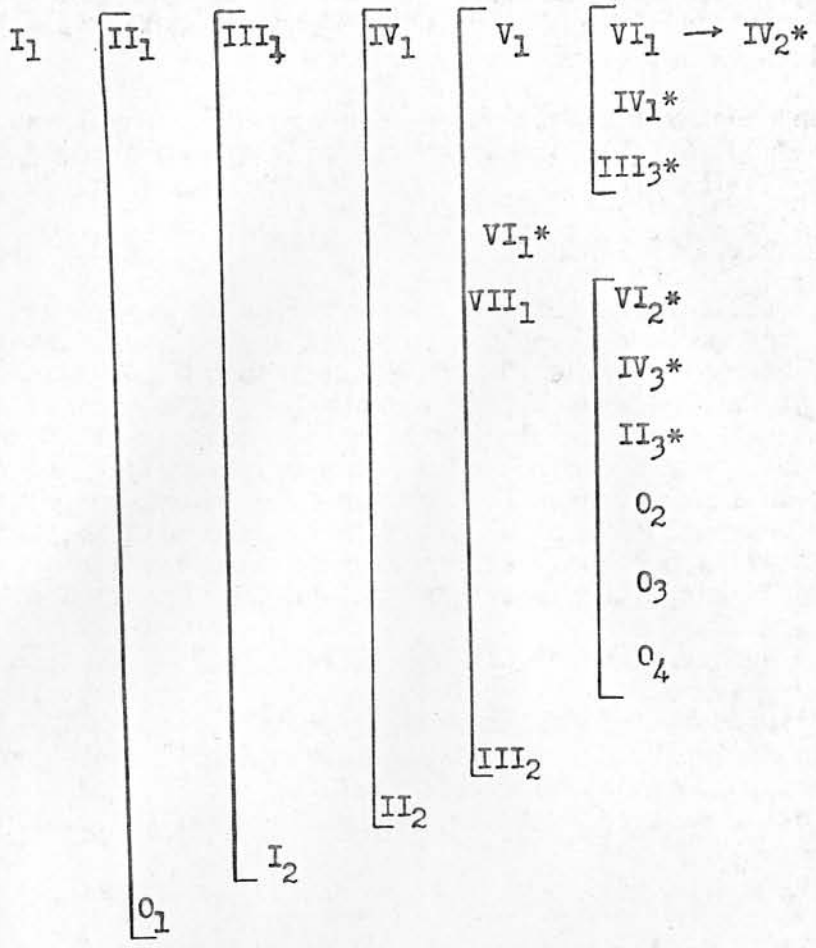
Division	1	2	3	4	5	6
Inputs						
I	Start	$II-O_2$				
II	$I-O_1$	$III-O_2$	$VII-O_2$			
III	$II-O_1$	$IV-O_4$	$V-O_3$			
IV	$III-O_1$ & $V-O_2$	$VI-O_1$	$VII-O_2$			
V	$IV-O_1$					
VI	$IV-O_2$ & $V-O_1$	$VII-O_1$				
VII	$IV-O_2$					
Outputs						
I	II_1	O_1				
II	III_1	I_2				
III	IV_1	II_2				
IV	V_1	VI_1	VII_1	III_2		
V	V_1	IV_1	III_3			
VI	IV_2					
VII	VI_2	IV_3	II_3	O_2	O_3	O_4

F. Main Synthesis

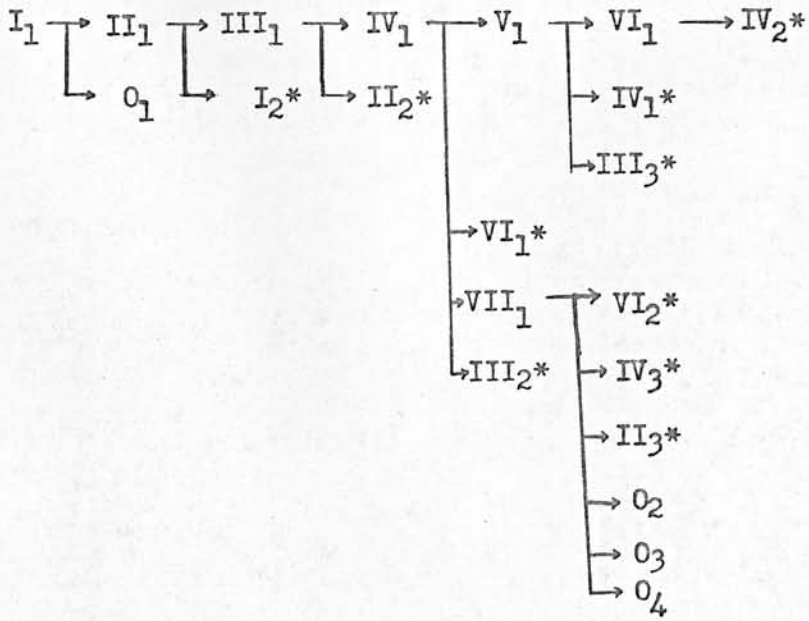
45. Table II, constructed in the last section, shows immediately how the complete program can be put together from those of the divisions, provided they satisfy the requirements in Section E. In fact the formula for the complete calculation in the notation of Reference (a), par. 108 is as follows:

$$I_1 \rightarrow :\!:\!. \; II_1 \rightarrow :\!:\; III_1 \rightarrow :\!. \; IV_1 \rightarrow :\; V_1 \rightarrow . \; VI_1 \rightarrow IV_2 *\,.$$
$$.\,\&\; IV_1 *\,.\,\&\,.\; III_3 *\,:\,\&\,:\; VI_1 *\,:\,\&\,:\; VII_1 \rightarrow . \; VI_2 *\;\&\; IV_3 *$$
$$\&\; II_3 *\;\&\; O_2\;\&\; O_3\;\&\; O_4\;:\,\&\,:\; III_2 *\,:\,.\,\&\; II_2 *\,:\,: \quad (23)$$
$$\&\; I_2 *\,:\,:\,.\,\&\; O_1$$

In the notation of Reference (a), par. 109 the program is as follows:



For some purposes the following is more compact.



(24)

This modification of the notation of Reference (a), par. 109 will be used hereafter. The star is used, instead of angular brackets, to indicate that the division has already occurred. Otherwise the notation is believed to be self-explanatory.

46. The general structure of the program is probably more perspicuous from Table II then from any of these notations. A flow chart is exhibited in Chart A.

G. Schedule of Subdivisions

47. For the purposes of later work we shall need a more detailed schedule of the problem than that given in Section E. That is, it will be necessary to divide the divisions into smaller units, called subdivisions, each of which performs some definite part of the computation. These subdivisions will be so chosen so that the more important ones contain schemes of $\mathcal{L}_2$ computation which form a unit in some sense, so that programs for them as independent computations may be presumed to be available, or else they are to be modifications of such schemes. It will, however, be necessary to make certain transitional computations of a relatively trivial nature into independent subdivisions.

48. The Subdivisions of divisions I, II, III, and, to some extent, also IV were given above in Section E. Here we shall specify in more detail what is to be done in these subdivisions, and shall also set up subdivisions for V, VI, and VII. Certain details in regard to inputs and outputs cannot be given until Chapter IV, and the synthesis of the composite program from the subdivisions will be considered there.

I. Main Control

A. Main Start

Receive the start signal. Puts $\lambda = 0$. Makes other initial adjustments, and may contain a check that $k > p + q$.

B. Group Iteration Control

Contains an output to II A, and an open output which indicates normal completion of the computation.

II. Group Control

A. Group Start

Receives starting signal from I-O_1. Prints $\phi = \phi_0 + \lambda \Delta \phi$ to indicate starting of a new group. Sets $\alpha = \alpha_0$, $\mu = 0$, $\rho = \rho_0$, $\kappa = p$. Clears group error register.

B. Round Iteration Control (see Section E)

C. Group Termination

Prints the group error register. Output to I_2.

D. Abnormal Group Termination

Receives signal from VII - O_3. Puts a 1 in group error register and goes out to C.

III. Round Control

A. Round Start

Receives signal from II-O_1. Prints $a = a_0 + \rho \Delta a$. Clears round error register. Sets $\mu = 0$.

B. Phase Iteration Control

C. Round Termination

Prints round error register and goes out to II_2.

D. Abnormal Round Termination

Receives signal from $\overline{V}$ - O_3. Puts 1 in round error register and goes out to C.

IV. Phase Control

A. Phase Start

This is a more complicated program than in the preceding cases, since it is necessary to set up the working locations. It comprises three main parts:

1. Reading $F_{\mu,\kappa,\lambda}$ into E_i.
2. Iteration control of 1 from $i = -f$ to $i = f$ inclusive.
3. A discrimination as to whether $\mu > 0$ or not, with outputs to IV - O_1 and IV - O_2.

B. Phase Termination Test

Receives from VI - O_2 signal that calculation is completed. Discriminates as to whether $\mu \geqq 0$. If $\mu > 0$ goes to C, if $\mu = 0$ to $IV - O_3$

C. Phase Termination

Prints ν and goes out to III_2.

D. Special Termination for $\mu = 0$.

Receives signal from VII - O_2 that iteration is complete. Puts $x_0 + (\kappa + u)\Delta x$ in v. Prints u and κ (to serve as error signals for irregular calculations). Goes out to C.

V. State Determination

A. State Bracket Test

Distinguishes between the following alternatives with outputs as indicated.

$$\left.\begin{aligned} v_{-1} \leq a &< v_0 \\ v_0 \leq a &< v_1 \\ v_1 \leq a, \quad v_2 &\leq v_1 \end{aligned}\right\} \quad \text{out to V - } O_1$$

$$v_1 \leq a, \quad v_1 < v_2 - \quad \text{Out to B.}$$

B. State Iteration Control

Controls changes of κ. If λ is not too large, puts $\kappa+1$ for κ, sets $u=0$ and goes out on V - O_2. If κ is too large, goes out on V - O_1.

VI. Calculations

A. Constant Part (Coefficients, etc.)

Sets $\nu = 0$. Contains input VI_1, and goes out to B.

B. Variable Part

Receives from A and also from VII - O_1. Goes out to IV_2.

VII. Inversion Control

A. First Transition

Puts $-c(v-a)$ for v.

B. Approximation Tests

Distinguishes the following alternatives with outputs as indicated:

$v \leqq -\epsilon$ out to VII - O_4

$|v| < \epsilon$ out to VII - O_2

$v \geqq \epsilon$ out to C.

C. Iteration Control

Prevents ν from getting too large. If $\nu > n$ goes out on VII - O_5, otherwise goes to D.

D. Second Transition

Sets $u + v$ for u.

E. Bound Test

Distinguishes the alternatives

$u < -p$ out to VII - O_6

$-p \leqq u \leqq q$ out to VII - O_1

$u > q$ out to VII - O_3.

49. The subdivisions II A, II D, III A, III D, VII A, and VII D are comparatively trivial examples of arithmetic programs in the sense of Chapter III below. Since a general theory of arithmetic programs is developed there, the subdivisions mentioned will not be considered further. As for I A, it must contain provision for calculating f, g, and h, and for seeing that substitutions are made so that later orders can be executed correctly. This involves considerations of Chapter V. Likewise the subdivisions IV A, VI A, VI B, and the printing subdivisions I C, II C, III C, IV C, and IV D require considerations of Chapter V. The subdivisions I B, II B, III B, V B, and VII C are iteration controls; they are discussed in Chapter IV, Section C. Iteration controls are also involved in IV A, VI A, and VI B. The discrimination programs V A, VII B, and VII E are considered in Chapter IV, Section B. The final synthesis of the composite program from its subdivisions is taken up in Chapter IV, Section D, but certain details have to be left until Chapter V, Section E.

CHAPTER II

Basic Programs and Fundamental Principles

50. Evidently the simplest possible programs are those which consist of a single order, plus such outputs and data as are necessary to make a complete program. Such programs will be called basic programs. In order to form a secure foundation for programming theory, it is necessary to devote some attention to these programs. Evidently the listing of these programs is equivalent to listing the orders of the machine. The present chapter will be devoted to these programs. It will also contain certain conventions and principles which will be fundamental to the later chapters. In these general discussions we shall use symbols in senses which differ from those assigned to them in Chapter I, but in the applications to inverse interpolation the notation of Chapter I will be maintained. Confusion between these notations will not be serious, because it will generally be clear from the context which notation is intended.

A. General Preliminaries

51. In this section we shall introduce some general concepts and principles which are fundamental to the theory of programming developed below.

52. The first concept to be introduced is that of a locatum. Roughly speaking the locata are the words actually in the machine at a given time. But, as the introduction of a new technical term suggests, this rough definition is not sufficient; we are thinking of these words as tied to locations in a way which requires further definition. A locatum is neither a fixed location in the machine, nor a word in such a fixed location, nor a fixed sequence of digits. Rather, a locatum is a word occupying a position which is either the accumulator, the register, or a location in the memory which corresponds (by some transformation) to a fixed position in a program, called the reference program, which is given in advance. Thus, a locatum is a variable which can take different values in different stages of a computation and in similar computations based on different data; it may occupy different positions in different programs being compounded, and conceivably in different configurations of the same computation. We shall designate unspecified locata by the letters "λ" and "μ" the accumulator by "A," the register by "R," and locata in the memory by the letters "x," "y," "z," "u," "v," "w." With reference to any fixed configuration or program we may use these same symbols to designate the locations occupied:

53. The concept of locatum can be illuminated a little by reconsidering some points concerning program composition developed in

Reference (a). Let $\mathfrak{X}$ and $\mathfrak{Y}$ be two normal programs (Reference (a), par. 68). In Reference (a), par. 81, a program

$$\mathfrak{X} \longrightarrow \mathfrak{Y} \tag{25}$$

was defined when $\mathfrak{X}$ and $\mathfrak{Y}$ have the same quantity program. But if x and y have different quantity programs (25) is meaningless. For the case of greatest interest, where the quantity programs are blanks, it was shown in Reference (a), IV C that we could define a variety of substitutions of $\mathfrak{Y}$ for O_1 in $\mathfrak{X}$ by identifying locations in their quantity programs in various ways. This could be done by identifying these locations with those in a given quantity program, there called D. Thus we have to deal with locata with respect to this D as reference program. To pursue the matter further, suppose that $\mathfrak{X}$ and $\mathfrak{Y}$ have each four quantities; that the first quantity in $\mathfrak{X}$ and the second and fourth in $\mathfrak{Y}$ are to be identified, likewise the third and fourth in $\mathfrak{X}$ and the first in $\mathfrak{Y}$. Then the composite program would be represented, in the notation of Reference (a), par. 91, as

$$\mathfrak{X}\,(Z_1, Z_3, Z_2, Z_2) \longrightarrow \mathfrak{Y}\,(Z_2, Z_1, Z_4, Z_1)\,. \tag{26}$$

Here we have locata Z_1, Z_2, Z_3, Z_4 with respect to a reference program composed of these four quantities in some order. (Note that the exact arrangement of the reference program makes no difference.) For the inverse interpolation problem I D above gives a list of possible locata, viz., those in Table I, the $\mathfrak{x}_i$, $F_{\mu\kappa\lambda}$, $G_{\mu\rho\lambda}$, etc. These will occupy different places in the various subordinate programs, and many of them will take different values throughout the computation. In these examples the locata are variable quantities. That is typical for locata in the memory.

54. In the following, when we set up schedules for particular programs, for example, for the basic programs, we shall assign variables to the locations in the quantity program.* These variables will then appear in the symbol adopted for the program. When programs are combined, various locata with respect to some reference program will be substituted for these variables. This device will allow modes of composition to be indicated just as (26) indicates a composition of the original $\mathfrak{X}$ and $\mathfrak{Y}$ are thought of as $\mathfrak{X}\,(x_1, x_2, x_3, x_4)$ and $\mathfrak{Y}\,(x_1, x_2, x_3, x_4)$.

55. The second concept to be defined is that of a <u>term</u>. A term is a word which we can think of as constructed by the machine from its locata at any stage. The constants 0 and 1, which can be generated within the machine, together with the locata, will be designated as <u>basic terms</u>. We shall designate unspecified terms by the Greek letters

*These variables, of course, are then locata for that particular program as reference program.

ξ , η, ζ and sometimes ϕ, ψ, ω . The notion of term is necessarily a little vague, but special kinds of terms will be defined more precisely later.

56. It is sometimes necessary to represent terms as functions of other terms. For this purpose we need a notion akin to that of indeterminate in algebra. We shall use the letters s, t for such indeterminates. We may consider the indeterminates as adjoined to the basic terms. When this is done the narrower class of terms defined in the preceding paragraph will be called definite terms; a term in the extended sense will be called an indefinite term or a function. Generally the letters ϕ, ψ, ω will denote functions.

57. A term will be said to be independent of a location λ if it is not changed when λ is changed.*

58. The scheme of notation developed below is based on the principle that if ξ is a term and λ is a locatum

$$\{\xi : \lambda\} \tag{27}$$

shall designate a program which calculates the term ξ and stores it in the locatum λ , then comes out on a single output O_1; while if Φ is a predicate constructed by logical connectives from equalities and inequalities of terms,

$$\{\Phi\} \tag{28}$$

shall designate a discrimination problem which tests whether Φ is true and comes out on O_1 if it is, on O_2 if it is not. Certain programs of this character will be found among the basic programs; for certain others we shall adopt systematic definitions later.

59. In studying the composition of such programs the following principle will be frequently referred to. Let $\phi(x)$ be a function which depends on a set L of locata as parameters. Let ξ be a term such that a program for $\{\xi : \mu\}$ does not change any of the locata in L

---

*This has to be understood in a more rigorous sense with reference to a program (27) defined below. Such a program is independent of x if x is a dead location for (27). When (27) has been assigned to ξ , this definition can be extended to ξ itself.

Then the right side of the equation

$$\{\phi(\xi) : \lambda\} = \{\xi : \mu\} \longrightarrow \{\phi(\overset{\mu}{\lambda}) : \mu\} \tag{29}$$

will be a program which ends with $\phi(\xi)$ in λ. In that case, if the programs on the right have already been defined while that on the left has not, the equation (29) might be acceptable as a definition of the left side. This will not be the case if the locata in L are changed by $\{\xi : \mu\}$. Thus if $\phi(t) = A + t$, $\xi = \alpha$, $\lambda = \mu = A$, the left program ends with $A + \alpha$ in A, the right side with 2α in A.

60. It is assumed below that all programs are normal in the sense of Reference (a), par. 68. Further it will be supposed the quantity programs contain no dead locations, and that the locations belong to one or the other of the following three classes:

a. Datum locata - containing the given quantities of the problem. We shall assume these are not changed throughout the computations, i.e., they are programmed to send, but never to receive.

b. Answer locata. These are programmed to receive answers, but not to send.*

c. Auxiliary locata - which can be programmed both to receive and send. It is supposed that the initial contents of these locata have no effect on the answers, that they are changed, or at least may be changed during the computation, and that they are free for other uses when the computation is completed.

61. In writing out programs in detail, i.e., in giving the actual contents of the memory when the program is put on the machine, a three-column technique will be used. The first column gives the

---

*This is true for the computation as a whole. These answers are first calculated in part D of the program, then transferred to parts G or H. But for subordinate parts of the computation locata which are needed for later parts will be called answer locata even if they send. Thus the definition of answer locatum for a computation is determined by relation of the computation to some larger computation of which it forms a part.

location number,* the second the operator, and the third the datum number. The exit number will not be indicated; the exit in all cases is supposed to be the next word. Additional columns may be used for explanatory purposes; in fact, it will frequently help to indicate the basic program corresponding to each order.

B. General Considerations Regarding Basic Programs

62. The notion of basic program was defined in par. 50; and it was pointed out there that there is a one-to-one correspondence between basic programs and orders.** We shall consider in this chapter an enumeration of basic programs, and hence of the orders, for an idealized Institute of Advanced Study machine as specified in Reference (c) and in the Introduction. The result of our discussion will be the schedule given in Table III. In this section we shall bring forward certain a priori considerations regarding the construction of such a schedule.

63. The schedule in Table III is a modification of that given in Reference (c). Some of the changes are due to the idealizations here made; others are suggested by certain gaps in the scheme in Reference (c), viz., where symmetry, or some similar consideration, indicates orders not present there. Since, as the authors of Reference (c) themselves remark, the question of whether a program is to be made basic or composite is a matter to be decided only after consideration of various economic and other factors, and since there is room in the digits assigned to the operator for additional orders, it seems proper to include several such additions for at least tentative consideration. However, in the developments of the later chapters only a portion of the scheme in Table III will be used. This will be discussed below.

64. The basic programs may be divided into two main classes. The receptive programs are those in which the accumulator receives in the normal way specified in Section C, below, where the receptive programs will be discussed more fully. A nonreceptive program is one in which the accumulator acts in some other way - say, as transmitter (e.g., in $\{A:\alpha\}$), or as inert spectator, so to speak (e.g., in $\{\alpha:R\}$). The programs for multiplication and division are considered here as nonreceptive; these programs are really composite subroutines built into the machine, in that they consist of whole sequences of steps, in each of which the accumulator may function in different ways. It is convenient to class as nonreceptive certain irregular orders even though the accumulator receives.

65. Another classification of orders is that into those which do and those which do not involve the memory. By "memory" here, of course, the internal memory is meant; orders for getting words into and out of the machine are not considered here at all.

---

* These will begin always with location 1. cf. Reference (a).
** This fact makes it possible to use notations for basic programs and orders interchangeably. There is sometimes an advantage in doing so.

66. In the following there will be proposed a notation for basic programs. Along with this we shall need a code for the basic orders. Such a code was proposed in Reference (c). The symbol for an order consisted of two parts - a symbol for the operation and a place for the location number of the datum. For the idealized machine there should be a third column, for the exit number, but, as explained in the introduction, we shall not indicate the exit here. The code given by Reference (c) for the operators has been found to be confusing in certain cases; consequently it is modified here. The code for the initial orders is given in the column (b) of Table III.

C. Basic Functions and Receptive Programs

67. The following four functions will be called basic functions of the first kind:

$$\begin{aligned} \pi_0(x) &= +x \qquad \text{(or simply t)}^* \\ \pi_1(x) &= -x \\ \pi_2(x) &= +|x| \\ \pi_3(x) &= -|x|. \end{aligned} \tag{30}$$

68. The following four functions will be called basic functions of the second kind:

$$\begin{aligned} \pi_4(x) &= A + x \\ \pi_5(x) &= A - x \\ \pi_6(x) &= A + |x| \\ \pi_7(x) &= A - |x|. \end{aligned} \tag{31}$$

For these we have

$$\pi_{i+4}(x) = A + \pi_i(x) \qquad (i = 0, 1, 2, 3).$$

69. We consider now receptive programs of the form

$$\{\pi_i(\xi) : A\} \tag{32}$$

*It is convenient for certain purposes to write the + sign, in others not.

where ξ is a definite basic term.

70. These programs - or rather the corresponding orders - are all adopted as basic in Reference (c) when ξ is a locatum in the memory. Accordingly, these orders are taken as basic here.

71. For the other cases most of the orders are missing in the scheme of Reference (c). But the mechanism of these orders is probably this: there is a device for clearing the accumulator, and this device is operative for $i \leq 3$, inoperative for $i > 3$; there is a device for taking the complement, which is operative for i odd; and finally a device, operative in cases $i = 2, 3, 6, 7$, for testing the sign of ξ and throwing in the complement device when $\xi < 0$. Now these devices can be thrown in or out quite independently of one another, and without reference to the source from which the word entering the accumulator is derived. Presumably the wiring would be simpler if a 1 on a certain digit of the operator threw in the appropriate one of the above three devices regardless of what was in the rest of the operator, at least when some other fixed digit had the value 1.* Thus there would be no difficulty in wiring the gates so as to admit all these orders. Their utility and feasibility, however, is another matter; this merits some further discussion.

72. For $\xi = R$, Reference (c) admits (32) for $i = 0$ only. There would be no difficulty about carrying out such an order for all values of i, and there is some possibility that it may be useful. Therefore all eight cases are provided for in Table III.

73. In the case $\xi = A$, there is some doubt both as to the feasibility and the utility of the programs. For $i > 3$ the programs do not seem to be useful. For $i = 0$ the program has no effect. For $i = 2$ it is not clear whether there is a simple method of carrying out the order. But since, modulo 2, $-x = x+1$, the order could be carried out by adding a 1 into each digit and suppressing carries, then adding in a unit pulse on the right. For $i = 3, 4$ the program reduces to a discrimination plus that for $i = 2$. Thus Table III contains this program for $i = 2, 3, 4$.

74. In the case $\xi = 0$ it is assumed here that the program can be realized by ordering the accumulator to receive without sending anything in to it. The program, however, has no effect for $i > 3$, and merely clears the accumulator for $i = 0, 1, 2, 3$. Hence it is sufficient to take the case $i = 0$. This program enables one to clear the accumulator, and hence any position in the memory, without having to store a zero. The program appears to be useful, although it does not occur in Reference (c).

---

*To indicate a receptive order.

75. We consider now the case $\xi = 1$. Since the machine will generate unit pulses in the right most digit, and presumably will have to use such a "subtract correction pulse" in connection with other processes, it is assumed that the machine can deliver such a pulse without the necessity of storing it in the memory. Thus this, like the preceding, is a program without datum. The program is useful for $i = 0, 1, 4, 5$; it is particularly useful for $i = 4$ in connection with counting iterations. For $i = 2, 3, 6, 7$ it is a mere repetition of these other cases. Accordingly, this program is added to Table III for $i = 0, 1, 4, 5$.

76. In making a code for a program (32) the following scheme, different in some respects from that in Reference (c), is adopted here. As code for the operator for the initial order of (32) we shall write $\pi_i(c)$ (i.e. $+c$, $-c$, $|c|$, $-|c|$ respectively) for $i \leq 3$, and $\pi_{i-4}(h)$ for $i > 3$. (Here c suggests "clear", h, "hold".) The cases where ξ is not a position in the memory will be indicated by writing the appropriate one of the following four special symbols in the place for the datum.

A	for	$\xi = A$
R	for	$\xi = R$
U	for	$\xi = 1$
Z	for	$\xi = 0$

77. Besides these kinds of orders, an additional receptive order, not considered at all in Reference (c) appears to be useful. This order will read the location number of its own datum into the accumulator. When such an order receives control the location number will be available in the control register, and the order amounts to reading the datum number of the order in control into the accumulator. This order will be coded by writing Gc or Gh as operator, symbol, and the program will sometimes be indicated by writing d(*) for ξ in (27). The utility of this program will be discussed in Chapter V. It appears to be useful only for $i = 0, 4$, and indeed is excluded for the other values of i by the restrictions in Reference (a), Chapter II.*

D. Dependencies Among the Receptive Programs

78. The set of receptive programs, described in Part C above, is redundant in the sense that some of them have the same effect as composite programs made up from the others. Some of these dependencies will be stated here. In this $\phi(x)$ will be a function independent of A, and ξ is either 1 or a locatum in the memory or at R. We shall symbolize the fact that two programs X and Y have the same outputs with the same effect in each by $X \cong Y$.

---

*Alternatively we could read the entire order into A. In that case the objection based on Reference (a), Chapter II does not apply. The effect of the new order is then to make the location number of the order itself take the place of the datum number. But a new order is necessary in any case and the order described in the text seems more useful.

79. There are two kinds of dependencies to investigate, viz., those which require auxiliary memory - i.e., the use of one or more auxiliary** memory locata - and those for which no auxiliary memory is necessary.

80. Of these not involving auxiliary memory, we have

$$\{\phi(\xi):A\} \cong \{0:A\} \rightarrow \{A+\phi(\xi):A\}. \tag{33}$$

According to the criterion of par. 59 this will be true whenever ϕ and ξ satisfy the conditions stated, hence when $\phi(x) = \pi_i(x)$ for $i \leq 3$ (but not for $i > 3$). Again, we have

$$\{\phi(\xi):A\} \cong \{\xi:A\} \rightarrow \{\phi(A):A\}. \tag{34}$$

This eliminates the same programs as (33) on a different basis. We have also

$$\{A-\xi:A\} \cong \{-A:A\} \rightarrow \{A+\xi:A\} \rightarrow \{-A:A\}. \tag{35}$$

This reduces cases $i = 5, 7$ to $i = 4, 6$, respectively. In regard to the programs $\{\phi(A):A\}$ we have

$$\{|A|:A\} \cong \{A<0\} \begin{cases} \rightarrow \{-A:A\} \rightarrow O_1 \\ \rightarrow O_1^* \end{cases} \tag{36}$$

$$\{-|A|:A\} \cong \{A<0\} \begin{cases} \rightarrow O_1 \\ \rightarrow \{-A:A\} \rightarrow O_1^* \end{cases}.$$

Thus all the receptive programs of form (32) can be defined without auxiliary memory in terms of

---

** As defined in par. 60.

$$\{\ 0 : A\}$$

$$\{-A : A\}$$

$$\{A + \pi_i(\alpha) : A\} \qquad i = 0, 2$$

$$\{A + \pi_i(R) : A\} \qquad i = 0, 2$$

$$\{A + 1 : A\} \tag{37}$$

81. The second kind of dependency involves the use of an auxiliary memory location which we shall call ω. (If $\{A : R\}$ is present we might, under certain circumstances use R for ω.) Here we can eliminate $\{0 : A\}$ in either of the following ways.

$$\{0 : A\} \cong \{A : \omega\} \rightarrow \{A - \omega : A\}$$

$$\{0 : A\} \cong \{A : \omega\} \rightarrow \{-A : A\} \rightarrow \{A + \omega : A\} \tag{38}$$

We have also

$$\{A + \phi(\xi) : A\} \cong \{A : \omega\} \rightarrow \{\phi(\xi) : A\} \rightarrow \{A + \omega : A\} \tag{39}$$

Further

$$\{\phi(A) : A\} \cong \{A : \omega\} \rightarrow \{\phi(\omega) : A\} \tag{40}$$

As a consequence of these dependencies we can define all receptive basic programs accepted for Table III in terms of

$$\{A + \alpha : A\}$$

$$\{A + 1 : A\}$$

$$\{A - \alpha : A\} \quad \text{or} \quad \{-A : A\}$$

$$\{R : A\} \tag{41}$$

82. A further word is in order in regard to the programs for $i = 6, 7$. These are, of course, definable in terms of the others. But the definitions in terms of the orders admitted in Reference (c) are rather long. The definition of $\{A + |\xi| : A\}$, in fact, would be

$$\{A : \omega\} \to \{\xi : A\} \to \{A < 0\} \begin{cases} \to \{A - \omega : A\} \to \{A : \omega\} \to \{-\omega : A\} \to O_1 \\ \to \{A + \omega : A\} \to O_1^* \end{cases} \tag{42}$$

while that for $\{A - |\xi| : A\}$ would be obtained similarly. These programs are chiefly useful, presumably, in defining discriminations like

$$\{|\varkappa| < \psi\} \qquad \{\varkappa \neq 0\} .$$

The matter is gone into further in Chapter IV.

83. The dependencies in regard to the orders G are discussed in Chapter V.

E. Table of Basic Programs

84. For the nonreceptive basic orders we adopt the schedule in Reference (c), with some changes in coding and detail which will be clear from Table III.

85. The complete table of basic orders is given in Table III. The columns of this table have significance as follows:

a. Number: This gives each possible program a number. Note that for certain receptive programs π_i is involved, where i ranges order some of the numbers, 0, 1, 2, 3 as indicated. For the nonreceptive programs the number is indicated in the column for i = 0.

b. Initial Order: This gives the code for the initial order according to par. 76. Note the exit is given here for explicitness, although it will generally be omitted hereafter. On the use of A, R, U, Z in datum column, see par. 76.

c. GN Orders: These give, for comparison, the operational symbols for the corresponding orders in Reference (c). Note that the blank space for No. 11 arises because there is no operational symbol for this order in Reference (c). The entry a indicates the order is absent in Reference (c).

d. Characteristic: Here α is the number of orders, γ the number of quantities, τ the number of outputs.

e. Classifications: The 0's and 1's in these four columns have significance as follows:

	0	1
Reception	Non-receptive	Receptive
Datum	No datum	Datum in Memory
Clearance	Accumulator holds	Accumulator clears at start
Transmit	Accumulator not transmitting	Accumulator transmitting

These give suggestions for a possible digital coding of the basic orders. Note the last column only applies to non-receptive orders.

f. Effects: The last columns give the contents of the accumulator, register, and x locatum after completion of the program. A dash here indicates the locatum is not involved in the program.

85. The following concluding remarks relating to this schedule are now relevant:

a. The x in all these cases is the word in location 3. Except in numbers 15, 26, 34, 37 it is a quantity. It is to be used as a variable in connection with substitutions as indicated in par. 54.

b. Although orders have been included in Table III which do not occur in Reference (c), yet an attempt has been made in the schedules below to stick as closely to the schedule in Reference (c) as possible. The orders (32) where $\xi = 0$ or 1 have been assumed without further comment.* Where the additional orders are needed, this fact is noted and substitute actions in case the orders are not available are proposed. Certain of the orders are introduced in this way in Chapter III; in that case they are assumed without further comment thereafter.

---

*If these orders are not available the situation is the same as in par. 98 in Chapter III.

c. On the other hand the programs $\{\kappa\}$ and $\{\ell\}$, which do appear in Reference (c), are not considered here as independent orders at all. The need for these has not arisen because of our disregard of the position of the binary point. If these, and possibly some additional orders, are introduced, some modifications of the later chapters may be necessary.

d. Because of the insistence on monotone convergence, the orders involving absolute value are not needed in the problem of inverse interpolation of Chapter I. Although these orders are taken into account in the theory, they are used in the synthetic part of the argument only for variants.

e. The change in the operator symbols for control shifts was made because these symbols in Reference (c) have been found confusing. On the one hand capital and l.c. "C" are often confused in writing; on the other hand an unconditional control shift can be looked upon as a conditional control shift preceded by a clearing of the accumulator. Hence it has been decided to call the conditional one Kh , the unconditional Kc.

f. In the cases where the accumulator transmits it has been assumed that it retains its contents unchanged.

Table III

Basic Orders

Number for i = 0 1 2 3	Symbol	Initial order: Datum	Initial order: Exit	G - N Order for i = 0 1 2 3	Characteristic α	Characteristic y	Characteristic t	Classification: Reception	Classification: Datum	Classification: Clearance	Classification: Transient	Effects: A	Effects: R	Effects: X
1	$\{0 : A\}$	c Z	2	a	2	0	1	1	0	1		0	-	-
2 3	$\{\pi_i(1) : A\}$	π_i(c) U	2	a a	2	0	1	1	0	1		π_i (1)	-	-
4 5 6	$\{\pi_i(A) : A\}$	π_i(c) A	2	R a a	2	0	1	1	0	1	1	π_i (A)	-	-
7 8 9 10	$\{\pi_i(R) : A\}$	π_i(c) R	2	A a a a	2	0	1	1	0	1		π_i (R)	R	-
11 12 13 14	$\{\pi_i(x) : A\}$	π_i(c) 3	2	- M -M	2	1	1	1	1	1		π_i (x)	-	x
15	$\{d(*) : A\}$	Gc 3	2	a	3	0	2	1	1	1		d(*)	-	x
16 17	$\{A + \pi_i(1) : A\}$	π_i(h) U	2	a a	2	0	1	1	0	0		A+ π_i(1)	-	-
18 19 20 21	$\{A + \pi_i(R) : A\}$	π_i(h) R	2	a a a a	2	0	1	1	0	0		A+ π_i(R)	R	-
22 23 24 25	$\{A + \pi_i(x) : A\}$	π_i(h) 3	2	h -h Mh -Mh	2	1	1	1	1	0		A+ π_i(x)	-	x
26	$\{A + d(x) : A\}$	Gh 3	2	a	3	0	2	1	1	0		A+d (*)	-	x
27	$\{\sim\}$	r -	2	R	2	0	1	0	0	0		r(A)		-
28	$\{\ell\}$	L -	2	L	2	0	1	0	0	0		L(A)		-
29	$\{\sim R : A\}$	X 3	2	X	2	1	1	0	1	1		Note 1	Note 1	x
30	$\{A : R\}$	R A	2	a	2	0	1	0	0	0	1	A	A	-
31	$\{\sim : R\}$	R 3	2	R	2	1	1	0	1	0		A	-	A
32	$\{A/\sim : R\}$	* 3	2	*	2	1	1	0	1	0		A	A	x
33	$\{A : \sim\}$	S 3	2	S	2	1	1	0	1	0		A	-	A
34	$\{A : d(\sim)\}$	Sd 3	2	Sp	3	0	2	0	1	0		A	-	Note 2
35	$\{A : e(\sim)\}$	Se 3	2	a	3	0	2	0	1	0		A	-	Note 2
36	$\{K\}$	Kc 2	-	C	2	0	1	0	1	1		-	-	-
37	$\{A < 0\}$	Kh 3	2	Ce	3	0	2	0	1	0		-	-	-
38	stop	0 -	-	a	1	0	1	0	0	0		-	-	-

Notes 1. After multiplication A and R hold different digits of product. We assume here product is in A.
2. In these orders x has its datum or exit number changed. The order Se is not used hereafter.

CHAPTER III

Arithmetic Programs

87. With this chapter we begin the systematic synthesis of composite programs from the basic programs. The first type of programs to be considered will be that of a purely straightforward arithmetic character, without control orders (i.e., orders of types A, B, and D of Reference (a), par. 23*) or discriminations. A more or less systematic theory of such programs will be given, so that whenever the ξ in (27) is an arithmetic term in the sense defined below, an essentially unique program can be given which can be taken as the definition of (27).

A. Definitions

88. In this section we shall give formal definitions of certain kinds of terms, functions, etc. The fundamental definitions will be by induction. Such a definition involves two parts, viz., (a) specification of certain initial elements, and (b) specification that terms constructed from terms already in the class by certain specified constructions shall belong to the class. It is then to be understood, without further explanation, that the class consists of those terms which can be constructed from the initial elements by a series of the constructions indicated. The part (b) of such a definition will be called the induction.

89. The class of basic terms was defined in Chapter II, Section A (par. 55).

90. The class of arithmetic terms is defined as follows. Let L be a class of locata, and let $t_1, t_2, \ldots, t_n$ be indeterminates. Then ζ is an arithmetic term based on L, $t_1, t_2, \ldots, t_n$, if

a. Initial terms - one of the following holds:

$$\begin{aligned} \zeta &= 0 \\ \zeta &= 1 \\ \zeta &= t_i \qquad i = 1, 2, \ldots, n \\ \zeta &= \lambda \quad \text{where } \lambda \text{ is in L.} \end{aligned}$$

b. Induction - ξ and η are arithmetic terms, and one of the following holds:

$$\begin{aligned} \zeta &= \pi_i(\xi) \qquad i = 1, 2, 3. \\ \zeta &= \xi + \eta \\ \zeta &= \xi\eta \\ \zeta &= \xi/\eta \end{aligned}$$

*These include orders Gc and Gh.

91. Generally the class L will be left to the context; when not specified it will be the class of all locata.

92. An arithmetic function based on a class L of locata will be a function $\phi(t_1, t_2, \ldots, t_n)$ which is defined by an equation:

$$\phi(t_1, t_2, \ldots, t_n) = \zeta,$$

where ζ is an arithmetic term based on L, $t_1, \ldots, t_n$.

93. The class of elementary terms based on a class L of locata and indeterminates $t_1, t_2, \ldots, t_n$ is defined by induction as follows: ζ is an elementary term if

(a) Initial terms - same as for arithmetic terms.
(b) Induction - one of the following holds, where ξ is elementary and x is a location in the memory and in L:

$$\zeta = \pi_i(\xi) \qquad i = 1, 2, 3$$
$$\zeta = \xi + \pi_i(x)$$
$$\zeta = \xi + \pi_i(1) \qquad i = 0, 1, 2, 3$$
$$\zeta = \prime\xi$$
$$\zeta = \xi/\prime$$

94. An elementary function based on L is a function $\phi(t_1, \ldots, t_n)$ defined by an equation

$$\phi(t_1, t_2, \ldots, t_n) = \zeta,$$

in which ζ is an elementary term based on L, $t_1, t_2, \ldots, t_n$.

95. In all the above cases we shall say that a term or function depends on a basic term ξ if it fails to remain in the class when ξ is removed from L; otherwise it is independent of ξ.

96. The order of a term of these kinds is the number of times the induction process has to be applied. Thus if ξ is of order n, $\pi_i(\xi), \xi + \pi_i(x) \ldots$ are of order n + 1; while if ξ and η are of orders m and n, $\xi + \eta, \xi\eta, \xi/\eta$ are of order m + n + 1.

B. Specification of Elementary Programs

97. The compositions of programs which will be considered here depend, in the last analysis, on the associative laws considered in Reference (a), Chapter IV, Section E, together with the principle (29). We shall use, in particular, the following four special cases of this principle

$$(\alpha_1) \qquad \{\zeta : \lambda\} = \{\zeta : A\} \rightarrow \{A : \lambda\}$$

$$(\alpha_2) \qquad \{\zeta : \lambda\} = \{\zeta : R\} \rightarrow \{R : \lambda\}$$

$$(\beta_1) \qquad \{\phi(\xi) : A\} = \{\xi : A\} \rightarrow \{\phi(A) : A\} \tag{43}$$

$$(\beta_2) \qquad \{\phi(\xi) : R\} = \{\xi : R\} \rightarrow \{\phi(R) : R\}$$

These equations are valid, in the sense that the right hand sides satisfy the requirements of the notation on the left, only under restrictive conditions. These conditions are, however, vacuous in the case of the formulas (α). In the case of the formulas (β), we shall employ it at first in cases where the $\{\xi : A\}$ and $\{\xi : R\}$ do not involve any locata in the memory, hence the conditions will be fulfilled whenever ϕ does not involve either A or R as a parameter. This happens, in particular, if

$$\phi(t) = \begin{cases} \pi_i(t) & (i = 1, 2, 3) \\ t + \pi_i(n) & (i = 0, 1, 2, 3) \\ t + \pi_i(1) & (i = 0, 1, 2, 3) \\ t \cdot n \\ t / n \end{cases} \tag{44}$$

On the other hand, note that the conditions do not necessarily apply for (β_1) if $\phi(t) = t + \pi_i(R)$, in particular if $\{\xi : A\}$ involves a multiplication.

98. In the specifications below we shall give, in effect, a unique definition of the program

$$\{\zeta : \lambda\} \tag{45}$$

whenever ζ is an arbitrary definite elementary term. The definition will be such that the program does not disturb any locations in the memory except, possibly, λ itself. But the definition requires the use of some of the additional basic receptive programs which were discussed in Chapter II. Without the additional orders which go with these programs, the statement would be obviously false, since even $\{A : R\}$ cannot be accomplished without disturbing the memory. Now the possibility

of making such programs without using an auxiliary memory is a great advantage to the programmer. Therefore, it is recommended that, if it is not practical to design the machine so as to allow these additional orders, then a position in the memory should be permanently set aside for making the reductions contemplated in Chapter II, Section D.

99. The definition of the program (43) is an inductive one of more complex character than those made in Section A above. The definition will be made first for ζ a basic term; then, for ζ not basic, the program will be defined in terms of one where ζ is of lower order or one which occurs earlier in the inductive process. In the statement below we shall suppose ξ is an elementary term of lower order, and x is a locatum in the memory. The definition involves several parts, as follows:

a. If ζ is a basic term,

1. The program is basic if either 1) $\lambda = A$, 2) $\zeta = A$, or 3) $\zeta = x$ and $\lambda = R$.

2. If program is not basic, reduce to the cases $\zeta = A$ and $\lambda = A$ by (α_1).

b. If $\zeta = \pi_i(\xi)$ $\quad (i = 1, 2, 3)$,

1. The program is basic if $\lambda = A$ and $\xi = A$, R, x, or 1.
2. If $\lambda = A$ and program is not basic, reduce to $\xi = A$ by (β_1).

3. If $\lambda \neq A$ reduce to $\lambda = A$ by (α_i).

c. If $\zeta = \xi + \pi_i(x)$ or $\zeta = \xi + \pi_i(1)$ $(i = 0, 1, 2, 3)$,

1. The program is basic if $\lambda = \xi = A$.

2. If $\lambda = A$, $\xi \neq A$, reduce to $\xi = A$ by (β_i).

3. If $\lambda \neq A$, reduce to $\lambda = A$ by (α_i).

d. If $\zeta = \nu\xi$,

1. The program is basic if $\lambda = A$, $\xi = R$.

2. If $\lambda = A$, $\xi \neq R$, reduce to $\xi = R$ by (β_2).

3. If $\lambda \neq A$, reduce to $\lambda = A$ by (α_i).

e. If $\zeta = \xi/\nu$,

1. The program is basic if $\lambda = R$ and $\xi = A$.

2. If $\lambda = R$, $\xi \neq A$ reduce to $\xi = A$ by (β_1).

3. If $\lambda \neq R$ reduce to $\lambda = R$ by (α_2).

100. On this definition we can now make the following observations:

a. Each of the reductions in b, c, d, part 3 gives a definition of the form

$$\{\phi(\xi) : \lambda\} = \{\xi : \mu_1\} \rightarrow \{\phi(\mu_1) : \mu_2\} \rightarrow \{\mu_2 : \lambda\}.$$

In part 2 of these same cases, the third component is missing. Of the three components the first has a ζ of lower order, the second is basic, and the third comes under case a.

b. It follows by induction on the order of ζ that the program so defined does not change any locata except A, R, and λ, and that λ is only involved in the transfer at the last step. Since none of the functions ϕ in the various cases 2 is dependent on either A or R, the condition for the applicability of (29) are verified. It follows that the program actually does put ζ into λ, and does so without disturbing any locatum, other than λ, in the memory.

c. If ζ is a given elementary term, and if all missing parantheses in the algebraic notation for ζ are supplied, so that it is clear in what order the operations are to be performed, then it is uniquely determined which of the cases a - e applies. There will be two choices for the ξ only in the cases $x + y$, xy; if we take, say the first one occurring, in such cases, then it follows that a unique program is defined. This is all that can be expected; naturally $\{x + (y + z) : A\}$ will have a different program from $\{(x + y) + z : A\}$.

d. The program requires, in addition to the orders mentioned in comment (b) to Table III, the basic programs (32) when ξ = A or R and $\nu \leqq 3$. The orders for ξ = R and $\nu > 3$ are not needed. Further the proof of adequacy of the definition breaks down if $\zeta = \xi + \pi_\nu(R)$ is added to part (b) of the definition of an elementary term (par. 93).

101. As an example of the foregoing theory, we consider the program (45) where

$$\zeta = y_0 + x_1(y_1 + x_2(y_2 + x_3 y_3)).$$

Here we define

$$\eta_2 = y_2 + x_3 y_3$$

$$\eta_1 = y_1 + x_2 \eta_2$$

then

$$\zeta = y_0 + x_1 \eta_1 .$$

It is clear that ζ must come under case (c) of par. 99; hence by (c_3)

$$\{\zeta : \lambda\} = \{\zeta : A\} \rightarrow \{A : \lambda\}$$

By (c_2)

$$\{\zeta : A\} = \{x_1 \eta_1 : A\} \rightarrow \{A + y_0 : A\} .$$

Then by (d_2)

$$\{x_1 \eta_1 : A\} = \{\eta_1 : R\} \rightarrow \{x_1 R : A\} .$$

Combining the last three equations

$$\{\zeta : \lambda\} = \{\eta_1 : R\} \rightarrow \{x_1 R : A\} \rightarrow \{A + y_0 : A\} \rightarrow \{A : \lambda\} .$$

Similarly, with R for λ

$$\{\eta_1 : R\} = \{\eta_2 : R\} \rightarrow \{x_2 R : A\} \rightarrow \{A + y_1 : A\} \rightarrow \{A : R\}$$

$$\{\eta_2 : R\} = \{y_3 : R\} \rightarrow \{x_3 R : A\} \rightarrow \{A + y_2 : A\} \rightarrow \{A : R\} .$$

The final program, obtained by combining the last three, is exhibited in code form in Table IV for the case where λ is a position in the memory. Note that the program can be constructed, although it would be pedantic to go through the details, by the technique of composition developed in Reference (a). The last two columns in Table IV give the contents of A and R at various stages.

Table IV

Programs for $\{ y_0 + x_1 (y_1 + x_2 (y_2 + x_3 y_3)) : A \}$

	Explanation	Op	Datum	Contents of A	Contents of R
1	$\{ y_3 : R \}$	R	18	-	y_3
2	$\{ x_3 R : A \}$	X	14	$x_3 y_3$	-
3	$\{ A + y_2 : A \}$	+h	17	η_2	
4	$\{ A : R \}$	R	A		η_2
5	$\{ x_2 R : A \}$	X	13	$x_2 \eta_2$	-
6	$\{ A + y_1 : A \}$	+h	16	η_1	
7	$\{ A : R \}$	R	A		η_1
8	$\{ x_1 R : A \}$	X	12	$x_1 \eta_1$	
9	$\{ A + y_0 : A \}$	+h	15		
10	$\{ A : \lambda \}$	S	19		
11	Stop	0			
12	x_1				
13	x_2				
14	x_3				
15	y_0				
16	y_1				
17	y_2				
18	y_3				
19	λ				

C. The General Arithmetic Program

102. In order to construct programs (45) for arithmetic terms ζ which are not elementary, it will be necessary to use auxiliary memory. We shall call a program (45) involving m auxiliary memory locata an arithmetic program of degree m; in that case ζ will be called an arithmetic term of degree m.

103. To define such a program we proceed by induction. If ζ is elementary we use the definition of Part B; ζ is then, by definition, of degree 0. If ζ is not elementary we reduce, if necessary, to the case $\lambda = A$ by (α_1) of (43). We shall suppose $\zeta = \phi(\xi)$ where $\phi(x)$ is a function independent of A and R. There are two cases, $\xi \neq A$ and $\xi = A$.

104. If $\xi \neq A$ we use (β_1) of (43), viz.,

$$\{\phi(\xi) : A\} = \{\xi : A\} \rightarrow \{\phi(A) : A\} .$$

Suppose such a definition is valid, and that the two components on the right have degrees m and n, respectively. Then the m auxiliary locata of the first component on the right can be chosen so that $\phi(x)$ is independent of them and the n auxiliary locata of the second component so that as many of them as possible are among the first m. Then the composite program will have a degree which is the larger of m and n. Thus (α_2) gives an analysis into two components, neither of which has a higher degree than the program to be analyzed, and such that in one of them $\xi = A$, and the other has a simpler structure.

105. Consider now the case $\xi = A$. If the program is not elementary $\phi(x)$ will be of the form

$$\phi(x) \cdot \psi(x, \eta)$$

where η has to be calculated. To carry out such a calculation we have to clear the accumulator. This means that the contents of A must be moved to an auxiliary location ω. Then we can define

$$\{\phi(A) : A\} = \{A : \omega\} \rightarrow \{\phi(\omega) : A\} .$$

Here, since ω is a datum locatum for $\{\phi(\omega) : A\}$ it cannot be an auxiliary locatum; hence if $\{\phi(\omega) : A\}$ is of degree m, $\zeta = \phi(A)$ is of degree m + 1.*

*This conclusion depends on the restriction as to locata adopted in par. 60. If we were to admit that after a program had received from a datum location it could use that same location for auxiliary purposes, we should have difficulty as to the definitions of degree.

106. This process can evidently be carried through in all cases in which ζ does not depend on R. For every nonbasic arithmetic term is of one of the forms $\pi_i(\xi)$ or $\psi(\xi,\eta)$, where $\psi(x,y) = x+y, xy,$ or x/y. Here neither ξ nor η involves R if ζ does not.* If η or ξ does not involve A, then ζ is of the form $\phi(\xi)$ or $\phi(\eta)$ as required. If, in the last three cases, both ξ and η involve A, then let $\xi = \omega_1(A)$, $\eta = \omega_2(A)$, $\phi(x) = \psi(\omega_1(x), \omega_2(x))$, and we have $\zeta = \phi(A)$. Of course the cases where ζ is elementary do not interest us.

107. Since the process defines the program (45) in terms of similar programs where ζ has a simpler structure, we have an inductive scheme which must terminate by giving us a definition of (45) in terms of the basic programs. Of course, the definition is not uniquely determined by the ordinary algebraic expression for ζ, since the analysis into functions ϕ, ψ, etc., may be carried out in more than one way, and the auxiliary locata are arbitrary. But, when such an analysis is given, and an agreement is reached in the way of choosing the auxiliary locata, one and only one program is defined. The component programs in each stage of the analysis are of degree not higher than that of the composite programs.

108. Let us apply this process to the example

$$\zeta = (x+1)(y+1)(z+1). \tag{46}$$

Let $\xi_1 = x+1$ $\quad\quad$ $\eta_1 = y+1$

$\xi_2 = \xi_1\eta_1$ $\quad\quad$ $\eta_2 = z+1$

so that $\zeta = \xi_2\eta_2$

Then $\{\zeta : A\} = \{\xi_2 : A\} \longrightarrow \{A\eta_2 : A\}$

$\{\xi_2 : A\} = \{\xi_1 : A\} \longrightarrow \{A\eta_1 : A\}$

Since ξ_1 is elementary, we have

$$\{\xi_1 : A\} = \{x : A\} \longrightarrow \{A+1 : A\}.$$

*It must be clear that R is involved in a term if it appears in any component of it, even if it should cancel out. cf. footnote to par. 60.

On the other hand

$$\{A\,\eta_2 : A\} = \{A : \omega\} \longrightarrow \{\omega\,\eta_2 : A\}$$
$$= \{A : \omega\} \longrightarrow \{\eta_2 : A\} \longrightarrow \{A : R\} \longrightarrow \{\omega R : A\}\ ,$$

with a similar scheme for $\{A\,\eta_1 : A\}$. If we put these equations all together and expand the $\{\eta_2 : A\}$ and $\{\eta_1 : A\}$, we get the following program involving 12 orders (and a stop order):

$$\{x : A\} \to \{A+1 : A\} \to \{A : \omega\} \to \{y : A\}$$
$$\longrightarrow \{A+1 : A\} \to \{A : R\} \to \{\omega R : A\} \to \{A : \omega\} \qquad (47)$$
$$\longrightarrow \{z : A\} \to \{A+1 : A\} \to \{A : R\} \to \{\omega R : A\}\ .$$

109. This, however, is not the shortest program for calculating (46). In fact, set

$$\xi = xy + x + y + 1$$

$$\phi(t) = t^2 + t\ ,$$

then

$$\zeta = \phi(\xi)\ .$$

Hence our program is

$$\{\xi : A\} \to \{A : \omega\} \to \{\omega z + \omega : A\}\ .$$

The first and last components are elementary. The theory of elementary programs gives us

$$\{\xi : A\} = \{x : R\} \to \{yR : A\} \to \{A + x : A\} \to \{A + y : A\} \to \{A+1 : A\}$$

$$\{\omega z + \omega : A\} = \{z : R\} \to \{\omega R : A\} \to \{A + \omega : A\}\ .$$

So that we get the following program involving only nine orders:

$$\{x : R\} \to \{yR : A\} \to \{A + x : A\} \to \{A + y : A\} \to \{A+1 : A\} \qquad (48)$$
$$\to \{A : \omega\} \to \{z : R\} \to \{\omega R : A\} \to \{A + \omega : A\}\ .$$

110. This example suggests a modification in the definition of arithmetic term. From the point of view stressed in Chapter II, par. 55, a term should be thought of as the result of a process of construction from the locata. Different processes of construction should be thought of as different terms, even though the results are algebraically equal. In other words, the leading to (48) is a different term from that leading to (47). From such a point of view, a term has a unique program. But if we adopt that point of view, the leading to (48) is not an arithmetic term at all. What sort of modification of the definition is suitable from this standpoint is a question which it is not feasible for consideration here. The present standpoint has been that a term is an algebraic expression, that we indicate a process of construction by multiplying out, putting in parentheses, etc. so that by the rules of association to the left and of taking the left most element in case of doubt, the process of construction is suggested by the algebraic expression. From this standpoint the program (48) is not an instance of (45). It can be represented most compactly in the form

$$\{xy + x + y + 1 : w'\} \rightarrow \{z\,w + w : A\}\ . \qquad (49)$$

On the other hand, the program

$$\{(xy + x + y + 1)(z + 1) : A\}\ . \qquad (50)$$

is a ten-order program which differs from (48) in that the last three components are replaced by

$$\{z : A\} \rightarrow \{A + 1 : A\} \rightarrow \{A : R\} \rightarrow \{w\,R : A\}\ .$$

111. There is another lack of uniqueness in (45) in that the auxiliary locata are arbitrary. When the program is combined with other programs these locata have to be assigned to places in the reference program just like any others. They should, therefore, appear in the symbol for the program. The notation (45) would then have to be replaced by a more explicit notation, such as

$$\{\zeta : \lambda\}\ (w_1, w_2, \ldots, w_m)\ , \qquad (51)$$

in which $w_1, \ldots, w_m$ are the auxiliary locata.

112. An alternative procedure would be to set aside certain positions in the memory for this purpose. It has already been recommended that this be done if certain programs here taken as basic have to be programmed with an auxiliary memory. (This is to save the nuisance of having to keep track of this locatum.) Computations of considerable generality can be obtained with programs of comparatively small degree. Thus an arbitrary polynomial (with fixed n) in the form

$$a_o + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

is of degree 1; in the form of the Horner algorithm,

$$a_o + x\,(a_1 + x\,(a_2 + \cdots + x\,(a_{n-1} + x a_n)\cdots)),$$

it is elementary.

113. Another modification of the notation (45), viz.,

$$\{\zeta : \lambda_1, \lambda_2, \ldots, \lambda_n\}, \tag{52}$$

will be useful later. It denotes the program for putting simultaneously in the locations $\lambda_1, \lambda_2, \ldots, \lambda_n$. It may be defined thus

$$\{\zeta : \lambda_1, \ldots, \lambda_n\} = \{\zeta : A\} \rightarrow \{A : \lambda_1\} \rightarrow \{A : \lambda_2\} \rightarrow \cdots \rightarrow \{A : \lambda_n\}.$$

Note that it automatically puts ζ into A at the same time.

114. Subject to the explanations made in the last few paragraphs, (45) and its various modifications designate a uniquely determined program. As (48) shows it may not be the shortest conceivable program for a given algebraic expression. But its construction is a matter of routine. It could be constructed by a machine.

CHAPTER IV

Discrimination Programs

115. In this chapter we consider programs which involve, in addition to orders permitted in Chapter III, the discrimination order Kh. In order to synthesize orders of this sort we shall need also the order Kc. The treatment here will not be systematic, as in Chapter III. Rather the idea will be to discuss special cases which will be useful in the inverse interpolation problems, and to end up with a somewhat more detailed synthesis of the inverse interpolation problem than was possible in Chapter I. The essential new factor is the branching of the program caused by the discrimination.

A. Preliminaries

116. The basic discrimination program is $\{A < 0\}$. Written out in full, according to the explanations in Chapter II, this program is as follows:

1. Kh 3
2. 0 $(= 0_1)$
3. 0 $(= 0_2)$

The effect is to come out on 0_1 if $A < 0$, on 0_2 if $A \geq 0$. Note that 0_1 is necessarily the next word after the Kh order. In the idealized situation of Reference (a), where the exit of every order is explicitly indicated, there is no essential difference between 0_1 and 0_2, but when the exit is taken to be the next order automatically, there is an essential difference - the consecutivity between the discrimination and 0_1 must be maintained.

117. The requirement that the exit be the next order thus plays an essential role in connection with discrimination. In the programs of Chapter III, the orders are in a simple linear series, and the compositions used there are such as to maintain the consecutivity in the ordinary course of things. But with discrimination programs, it is necessary to consider explicitly the effect of the new convention. Evidently, in order that transformation be homomorphic, there is an additional condition besides the difference condition. This condition, which we shall call the consecutivity condition, is that whenever an order has an exit, that exit after the transformation must remain in the next memory position. The transformations proposed in Reference (a), Chapter 4, actually satisfied this condition, and they were more complicated than they might otherwise have been because the condition was anticipated. If it is desired to make a transformation which violates this condition, or to clear a loop program, that can only be done by inserting a Kc order at that point.

118. We shall distinguish between two kinds of outputs to a program. A primary output is one which the consecutivity condition requires to follow immediately after some order in the program. A secondary output is one where this is not so. Thus the outputs O_1 and O_2 of the basic discrimination program differ in that O_1 is primary while O_2 is secondary. An arithmetic program has only one output, and that is primary.

119. Now let Φ be a propositional function of certain locata, say, for instance, $\xi < \eta$, where ξ, η are arithmetic terms. For such a Φ we may seek to define a program with two outputs, one primary and one secondary, such that the program comes out on the primary output when Φ is true and on the secondary output when Φ is false. Such a program we shall call a two-way discrimination. When Φ is $A < 0$, the notation for the basic program agrees with this convention. For certain derived programs we shall adopt definitions as follows

$$\begin{aligned} \{\xi < 0\} &= \{\xi : A\} \to \{A < 0\} \\ \{\xi > 0\} &= \{-\xi < 0\} \\ \{\xi < \eta\} &= \{\xi - \eta < 0\} \\ \{\xi > \eta\} &= \{\eta - \xi < 0\} . \end{aligned} \tag{53}$$

120. Now consider the logical combinations $\Phi_1 \wedge \Phi_2$ (i.e., Φ_1 and Φ_2 both true) and $\Phi_1 \vee \Phi_2$ (i.e., either Φ_1 or Φ_2 or both). The corresponding programs can be defined this:

$$\{\Phi_1 \wedge \Phi_2\} = \{\Phi_1\} \begin{cases} \to \{\Phi_2\} \begin{cases} \to O_1 \\ \to O_2 * \end{cases} \\ \to O_2 \end{cases} \tag{54}$$

$$\{\Phi_1 \vee \Phi_2\} = \{\Phi_1\} \begin{cases} \to O_1 \\ \to \{\Phi_2\} \begin{cases} \to O_1 * \\ \to O_2 \end{cases} \end{cases} \tag{55}$$

121. We discuss next the problem of designing a program for $\{\sim \Phi\}$, where $\sim$ is the sign of negation. Such a program we shall call a negation of $\{\Phi\}$. According to the meaning of (28), its negation should be the same program with the outputs interchanged. But there is no simple way to realize such a program in terms of our present schedule of orders. There are the following alternatives. In the first place we could adopt as negation of (28) the program

$$\{\Phi\} \begin{cases} \to \{K_2\} \to O_2 \\ \to O_1 \end{cases} \tag{56}$$

In such a program both outputs are secondary, and we should have to distinguish two kinds of discriminations: primary ones as above defined, secondary ones with both outputs secondary. The negation of a primary discrimination could then be accomplished as above with one additional order; the negation of a secondary discrimination merely requires an interchange of the two outputs. In the second place we could define

$$\{A \leqq 0\} = \{A - 1 < 0\}, \tag{57}$$

and adopt the analogues of the definitions (53) with $\leqq$ instead of $<$, $\geqq$ in place of $>$. Then $\{\sim\varphi\}$ can be defined by applying the rules of propositional logic. The situation here in regard to additional orders is complex; if the original discrimination is compounded from n discriminations without negation, n additional orders will be necessary. Another possibility is to adopt $\{A \leqq 0\}$ as a new basic order. This should be considered if the complexity of machine structure due to an additional basic order is compensated for by a more efficient use of the memory.

122. We shall not, however, go further along this line. One reason for this is that complex discrimination programs are likely to have more than two alternatives. Thus the program

$$\{\varphi_1\} \begin{cases} \to \{\varphi_2\} \begin{cases} \to O_1 \\ \to O_2 \end{cases} \\ \to O_3 \end{cases}$$

is a generalization of (54) and is likely to be more useful. Again, in complex discriminations, the various component discriminations are likely to have elements in common. Programs formed by following blindly the schemes considered here may, therefore, involve unnecessary repetition. It seems best, therefore, to consider the various kinds of complex discriminations used in the inverse interpolation problem individually, and to leave generalizations until afterwards.

123. Special consideration, however, must be given now to the discriminations

$$\{|\xi| < \eta\}$$
$$\{\xi \neq 0\}. \tag{58}$$

These discriminations may be defined simply in terms of the orders involving an absolute value, and indeed they appear to be the principal reason for introducing those orders in the first place. We shall consider

these under the supposition that ξ has just been calculated and is in A, while η is a quantity ϵ stored in a memory locatum. Then these programs could be defined thus

$$\{|A| < \epsilon\} = \{|A| : A\} \rightarrow \{A - \epsilon : A\} \rightarrow \{A < 0\}$$

$$\{A \neq 0\} = \{-|A| : A\} \rightarrow \{A < 0\} \ . \qquad (59)$$

Let us contrast with these the definitions which could be made without the absolute value orders. The first of them could be defined by the following scheme

$$\{A \leqq 0\} \rightarrow \{-A : A\} \rightarrow \{A - \epsilon : A\} \rightarrow \{A < 0\} \rightarrow \begin{matrix} O_1 \\ O_2 \end{matrix} \qquad (60)$$

in which we have indicated, in a way which is presumably self explanatory, that the secondary output of the first discrimination is to be the third order. This involves four basic programs, whereas the previous one involved three. As for the second program, we have the following two possible definitions,

$$\{A < 0\} \rightarrow \begin{matrix} O_1 \\ \{-A : A\} \rightarrow \{A < 0\} \rightarrow \begin{matrix} \{\kappa_c\} \rightarrow O_1 * \\ O_2 \end{matrix} \end{matrix}$$

$$\{A - 1 : A\} \rightarrow \{A < 0\} \rightarrow \begin{matrix} \{A + 1 : A\} \rightarrow \{A < 0\} \rightarrow \begin{matrix} O_1 \\ O_2 \end{matrix} \\ O_1 * \end{matrix} \ .$$

Each of these involves four basic programs. If we were to take $\{A \leqq 0\}$ as basic (cf., par. 121), we could have a definition with two basic programs, viz.,

$$\{A \leqq 0\} \rightarrow \begin{matrix} \{A < 0\} \rightarrow \begin{matrix} O_1 \\ O_2 \end{matrix} \\ O_1 * \end{matrix} \ .$$

124. In view of the situation adduced in the preceding paragraph, there is some doubt as to the wisdom of introducing these absolute value orders. If these are all the uses which they have, they do not save as much space in the memory as some of the extra orders which have been considered here. If a proper technique of program composition is developed, the order for $\{A \leqq 0\}$ may possibly be to do as much, both for simplifying programming and for economizing the use of the memory, as these absolute value orders.* (Cf. par. 139.)

*In case the ξ of (58) is in the memory, the definition in terms of absolute value, analogous to the first definition (59), saves two orders, instead of one, over the analogue of (60).

B. Betweenness and Bracket Tests

125. Let y and z be given quantities such that $y<z$. Then y and z divide the real line into three intervals. By a betweenness test we mean a discrimination program with three outputs which will distinguish the three possibilities for a third quantity x. In the following we shall suppose that these outputs are numbered to correspond with open intervals as follows

$$O_1 \qquad x < y$$

$$O_2 \qquad y < x < z$$

$$O_3 \qquad z < x \ .$$

With reference to the ends of these intervals there are four possibilities. The middle interval may be closed at either end, at neither end, or at both ends. With reference to these possibilities, the situation can then be uniquely described by giving the inequalities which lead to O_2. These inequalities will be called the type of the program. Thus a betweenness program of type $y \leqq x < z$ will be one which comes out on O_2 when these unequalities hold, on O_1 when $x<y$, and on O_3 when $x \geqq z$.

126. We consider here ways of realizing such betweenness programs by composites of two discriminations, one on the relative magnitudes of x and y, and one on the relative magnitudes of x and z. We shall allow, at first, not only discriminations based on $\{A < 0\}$, which we shall call positive discriminations, but also those based on $\{A \leqq 0\}$, which we shall call negative discriminations.

127. For each type there are eight ways of realizing the program in this manner. For the two discriminations may be performed in either order, i.e., we may discriminate on either x, y or x, z first. For each discrimination there are four possibilities, for x, y these are $\{x<y\}, \{x \leqq y\}, \{y<x\}, \{y \leqq x\}$. Of these two and only two are compatible with the type, and of these two, one is positive and one negative, one makes the output (O_1 or O_3) primary and the other makes it secondary. Thus if we specify either the sign (positive or negative) or the character (primary or secondary output O_1 or O_3) of each discrimination and the order of performing the discriminations there is a uniquely determined program. We shall designate this program by writing the type in braces and attaching superscripts according to the following schedule:

Table V

	First discrimination on	Sign of y-discrimination	Sign of z-discrimination
1	y	+	+
2	"	+	-
3	"	-	+
4	"	-	-
5	z	+	+
6	"	+	-
7	"	-	+
8	"	-	-

Thus $\{y \leqq x < z\}^3$ shall designate the program

$$\{y \leqq x\} \begin{cases} \to \{x < z\} \begin{cases} \to O_2 \\ \to O_3 \end{cases} \\ \to O_1 \end{cases} .$$

128. It will not be necessary to exhibit all 32 of these programs. We shall consider only those arising in the inverse interpolation problem.

129. In the State Bracket Test (VA), the fundamental test is a betweenness test of type:

$$x_0 \leqq a < x_1 .$$

If we require that both discriminations be positive, only cases 1 and 5 are possible. The programs in these cases are

$$\{a < x_0\} \begin{cases} \to O_1 \\ \to \{a < x_1\} \begin{cases} \to O_2 \\ \to O_3 \end{cases} \end{cases} , \tag{61}$$

$$\{a < x_1\} \begin{cases} \to \{a < x_0\} \begin{cases} \to O_1 \\ \to O_2 \end{cases} \\ \to O_3 \end{cases} . \tag{62}$$

In both of these programs O_1 is primary and O_3 is secondary, while O_2 is primary in the first and secondary in the second. Now in this problem the O_2 is the O_1 of State Discrimination in Chapter I. This initiates a calculation. But a signal to start a calculation also comes from IV, and in fact from the primary output of a discrimination $\{\mu > 0\}$. Hence it will save the necessity of a Kc order to make O_2 secondary. This indicates that

$$\{x_0 \leqq a < x_1\}^5$$

should be chosen.

130. If both positive and negative discriminations are equally available then the natural question to ask is whether we want the outputs O_1 and O_3 to be primary or secondary. Let us suppose first we have a simplified problem with no irregular calculations. Then O_1 signals an irregular termination of the round. This is a very simple program which may well be put in a part of the memory adjacent to the State Discrimination program. Thus we should want O_1 to be primary. This conclusion is strengthened because in the more elaborate calculation we need additional tests on O_1. As for O_3 we shall have to go to V B there. Hence we want both O_1 and O_3 to be primary. In that case the test on a and x_1 will have to be negative. A glance at Table V shows the possible cases are now 2 and 6. The programs for these are as follows:

$$\{a < x_0\} \begin{cases} \to O_1 \\ \to \{x_1 \leqq a\} \begin{cases} \to O_3 \\ \to O_2 \end{cases} \end{cases} \tag{63}$$

$$\{x_1 \leqq a\} \begin{cases} \to O_3 \\ \to \{a < x_0\} \begin{cases} \to O_1 \\ \to O_2 \end{cases} \end{cases} . \tag{64}$$

Both of these leave O_2 in secondary position, but the first one seems the more natural.

131. This discussion would complete the setup of the state-determination program - except for the control on k - for the simplified case. Let us now consider how the additional tests for the more elaborate calculation are to be grafted onto such a betweenness program. In doing so we recall that it is not necessary to program a warning signal for an irregular calculation, for the only purpose of such a signal would be to show (12) was not maintained, and that could be shown by printing u. Hence all we need to do is to go to O_2 on a secondary. In O_1, then, we first test $\{a < x_{-4}\}$; in the primary case we terminate the round as before. In the secondary case we must program an adjustment to VII. How this is to be done is a matter for Chapter V. For the present it suffices to say that there must be a program Y_2 for setting VII abnormally, and also a program Y_1 for resetting it to the normal. The latter program can be put at the beginning of the state determination, the output of Y_2 should go to O_2. Unfortunately that is primary. However, if we use programs (61) or (63) we can regard the start of the second discrimination as a second input, and we can take the output of Y_2 to this input on a primary. Since $a < x_0$ holds, we have a fortiori $a < x_1$. So much for O_1. As for O_3 we have a simple $x_1 < x_2$ with secondary output to O_2. Using the notation for outputs used in Chapter I, the final program in case (61) or (63) is used will be

$$Y_1 \rightarrow \{x_0 \leqq a < x_1\} \begin{cases} \rightarrow \{a < x_{-4}\} \begin{cases} \rightarrow O_3 \\ \rightarrow Y_2 \rightarrow I_2^* \end{cases} \\ \rightarrow O_1 \\ \rightarrow \{x_1 < x_2\} \begin{cases} \rightarrow O_2 \\ \rightarrow O_1^* \end{cases} \end{cases} \tag{65}$$

where I_2^* denotes the second input to the betweenness program. If (62) is used, I_2^* must be replaced by O_1^*, which means a Kc order.

132. In the program (65), O_1 will appear as primary output unless (63) is used. Hence a Kc order is necessary in synthesizing the program for all cases where only positive discriminations are used. But we could avoid this by using, instead of (61), the order

$$\{x_0 \leqq a \leqq x_1\}$$

whose program is similar to (61) except that the second discrimination is $\{x_1 < a\}$. The effect of this would be to allow a calculation for $u = 1$. This will likely do no harm; in fact any "ordinary" interpolation formula is exact at $u = 1$.

133. So much for State Determination. The various alternatives considered do not differ materially from one another. We shall take (65) in combination with (63), defining the negative discrimination involved by (57). The extra order necessary to use a negative discrimination turns out to be necessary anyhow. The State Bracket Test is set up by this method in Section D below.

134. In VII B we need a discrimination of the type

$$\{-\epsilon < \nu < \epsilon\} . \tag{66}$$

Inasmuch as $-\epsilon$ can be read directly from the memory, this is essentially of the type

$$\{y < \nu < z\} .$$

If we are confined to positive discriminations, the only two possibilities are

$$\{y < \nu\} \begin{cases} \to \{\nu < z\} \begin{cases} \to O_2 \\ \to O_3 \end{cases} \\ \to O_1 \end{cases}$$

$$\{\nu < z\} \begin{cases} \to \{y < \nu\} \begin{cases} \to O_2 \\ \to O_1 \end{cases} \\ \to O_3 \end{cases}$$

Hence the only program* for this test, which we shall call an approximation test, is essentially

$$\{-\nu - \epsilon < 0\} \begin{cases} \to \{\nu - \epsilon < 0\} \begin{cases} \to O_2 \\ \to O_3 \end{cases} \\ \to O_1 \end{cases} . \tag{67}$$

135. If negative discriminations are available, we could make a program in which O_1 and O_2 were secondary and O_3 primary by the program

$$\{-\nu - \epsilon < 0\} \begin{cases} \to \{\epsilon - \nu \leqq 0\} \begin{cases} \to O_3 \\ \to O_2 \end{cases} \\ \to O_1 \end{cases} . \tag{68}$$

*This program requires six active orders.

136. We can get the advantages of this sort of program by using only positive discriminations. If we make the test of type

$$-\epsilon < \nu \leqq \epsilon \ , \tag{69}$$

with program $\{-\nu - \epsilon < 0\} \rightarrow \{\epsilon - \nu < 0\} \rightarrow O_3$, O_1, O_2.

137. For the bounds test (VII E) $\{-\phi \leqq u \leqq \phi\}$ we require that O_2 be secondary (since it refers back to VI B), and it is an advantage to have O_3 primary. In the scheme below O_1 has been made secondary.* A suitable program is

$$\{-\phi \leqq u\} \rightarrow \{\phi < u\} \rightarrow O_3, \quad O_1, \quad O_2$$

138. In this discussion we have considered various possibilities for the three betweenness tests - viz., the "bracket test" (V A) in State Determination, the "approximation tests" (VII B) and "bound test" (VII E) in Inversion Control. In two of these cases we have found we can get exactly what we want without extra orders by using negative and positive discriminations. At the same time, we can get exactly what we want with positive discriminations only by using an extra order, or we can get a perhaps trivial modification of what we want without either extra orders or negative discriminations. We shall choose the middle one of these alternatives. That is, we assume for these tests

V A State Bracket Test - given by (65) and (63),
VII B Approximation Test - given by (67),
VII E Bound Test - given by (70).

139. We may note that if we identify O_1 and O_3 the approximation test becomes equivalent to

$$\{|\nu| < \epsilon\},$$

*This choice was made so that VII E would not break into detached parts. Actually a primary O_1 would save one order.

which can be made by orders involving absolute value as mentioned at the end of Section A. When ν is stored in the memory (68) involves then six active orders, the first program in (59) only three, while (60) would then require five. Thus the test to insure monotone convergence requires three extra orders besides those needed in connection with V A (see Chapter V). However, there are indications that the extra programming necessary to secure monotone convergence is not merely a matter of seeing how a more difficult situation can be taken care of. One can get very definite control over the accuracy of successive approximations by getting two monotone sequences which converge to the limiting value from opposite directions.* Thus in the present case, where we are subject to (7) and (8), the method of false position could give us, in addition to our sequence $\{u_n\}$ a second sequence $\{u'_n\}$ converging to the root of $P(u) = 0$ from above, and a discrimination

$$u'_n - u_n < \epsilon$$

would insure that our result was accurate to within ϵ. Thus the great importance of inequalities involving absolute values in the theory of functions may not be reflected in the numerical analysis of the future.

140. A betweenness test of a special kind, applicable for testing

$$y \leqq A \leqq y + z ,$$

is programmed in Chapter V, Section D. (See Table XVIII.)

C. Iteration Controls

141. We speak of an iteration when we have the following situation. We have a certain program (or "subroutine") Y depending on a parameter i. The composite program is to be so arranged that an output of Y may cause i to be increased and then order a repetition of Y with the increased i, this to be under the control of a discrimination which causes the process to terminate when i exceeds (or equals) a certain quantity m. Generally the increase in i each time Y is completed is an addition of 1, but by an easy generalization we can take care of the case where i is increased by an arbitrary increment δ, or even when more general operations are performed on i.

142. Iteration so defined is a very common process. Moreover, its various occurrences have a similar structure. Consequently, any

*As an example of how this can be done in the case of solutions of ordinary differential equations, see Reference (f).

collection of programs will have standard procedures for controlling iterations.

143. Any iteration involves three constituent programs. First there is the program Y which is to be iterated. This will be called the working program. Then there is a program X which precedes the iterative process; this will be called the preliminary program. Finally, there is a program Z, called here the terminal program, which is initiated when the discrimination signals the last iteration is complete.

144. An iteration control program is a program which can be combined with three such programs X, Y, Z so as to bring about an iterative process. It has two outputs: working output, O_y, which goes to Y, and a terminal output, O_z, which goes to Z. It must be capable of receiving a signal from an output of Y, and may or may not receive a signal from X. It must have a quantity program capable of storing m and i, and also δ if it is used.

145. We shall suppose that the operation to be performed on i at the end of each iteration is the addition of 1. Then the locatum i merely counts the iterations. The iteration is thus over a certain range of integers, from an initial value of i to a final value. The final value is m, or can be calculated from m, which is stored in the control program. On the other hand we shall suppose the initial value is set in X, so that the same control program can be used regardless of the initial value. We shall assume here that this initial value is 0, since the modifications are obvious if it is something else.

146. Iteration control programs may be classified as initial and final. An initial iteration control is one in which the first iteration is designed to go through the iteration control. Such a control program receives from X and makes a discrimination before proceeding to Y. Since the procedure for increasing i must come after Y in any event, such a control must contain two detached parts: a first part, preceding Y, which contains the discrimination, gives the output signals, and receives from either X or the second part; and a second part, following Y, which receives from Y, makes change in i, and goes back to the first part.* Here the relations of before and after refer to passage of the control from X through to Z. A final iteration control is one designed to come after Y in the above sense. The initial cycle is started from X; the iteration control receives an output from Y, decides whether a new cycle is to be ordered, and changes i if it is. Of course, an initial control can be used as final and vice versa. If an initial control is used as final, the initial cycle will be repeated; if a final control is used as initial, the iteration will start with i = 1. Generally a final control seems the superior procedure, but since the detached parts of an initial control illustrate some

*This is modified in the case of It_6 below, but the two detached parts are still necessary. The program It_6 is really a hybrid.

possibilities of program construction which the final ones do not, both kinds are considered here.

147. A second classification of iteration controls is into primary and secondary, according to whether the working output comes from the primary or secondary output of the discrimination. Generally one would suppose that initial programs were primary and final ones secondary, but provision is made for all four of the possible combinations. Uses will be found for programs which are both final and primary.

148. Outline programs of a set of six iteration control schemes are given in Table VI. The following remarks will explain the construction of this table. In the first 10 rows the orders are indicated by the symbols for the corresponding basic programs, without the braces. The outputs, however, will be indicated by O_y, O_z as above. A horizontal line of any kind indicates that the preceding order does not have an exit; the natural breaks in the programs occur at these points. A double horizontal line indicates that the program (Y) which is to be substituted for the preceding output must have a secondary output to carry the control back to the beginning.* A dotted horizontal line indicates that the following order is a secondary input of the program, and can receive from a primary output of a preceding Y. Thus a double line indicates that a Kc order may be necessary at the end of Y, and this may compensate for the Kc orders scheduled as part of the control program in the other cases. The remaining rows give various characteristics of the programs. A "0" in the row Ax indicates that the initial value of i (here 0) must be set in the accumulator when the control passes from X to order No. 1; a dash indicates the initial content of A is irrelevant. The rows "I or F" and "P or S" indicate whether the program is initial or final, primary or secondary, respectively. The row i_y indicates the contents of the i-locatum when the control reaches O_y; here an "i" without subscript indicates the value of i when the control reaches order No. 1. Evidently if the program in question is used initially, the value of i in the first cycle will be that obtained by giving i its initial value (viz., 0) in this row. The row i_f indicates the last value of i_y. The row i_z indicates the value of i when the control reaches O_z. The rows i_f and i_z give the values of i_f and i_z if the fundamental discrimination is $\{A \leqq 0\}$.

149. The program indicated in the kth column of Table VI will be referred to hereafter as

$$It_k\ (m,\ i).$$

---

*In the case of It_2 this is generally taken care of by putting Y before the input of the control program.

Table VI

Iteration Control Programs

	It_1	It_2	It_3	It_4	It_5	It_6
1	A : i	i : A	i : A	A : i	i : A	A-m:A
2	A-m:A	A+1:A	A+1:A	m : A	A-m:A	A <0
3	A<0	A : i	A : i	A-i:A	A<0	A+m:A
4	0_y	m : A	A-m:A	A<0	i : A	A+1:A
5	i : A	A-i:A	A<0	0_z	A+1:A	A : i
6	A+1:A	A<0	0_y	0_y	A : i	0_y
7	Kc (to 1)	0_z	0_z	i : A	0_y	i : A
8	0_z	0_y		A+1:A	0_z	Kc (to 1)
9				Kc(to 1)		0_z
10						
A_x	0	-	-	0	-	0
I or F	I	F	F	I	F	I
P or S	P	S	P	S	P	P
i_y	i	i+1	i+1	i	i+1	i+1
i_f	m-1	m	m-1	m	m	m
i_z	m	m+1	m	m+1	m	m
i'_f	m	m-1	m	m-1	m+1	m+1
i'_z	m+1	m	m+1	m	m+1	m+1

It will be supposed m and i are put in the first and second positions in the quantity program. When the third place in the quantity program is reserved for δ, and the order $A + 1 : A$ is replaced by $A + \delta : A$, the program will be designated

$$It_k (m, i, \delta) .$$

If ξ is an arithmetic term, the notations

$$It_k (\xi, i) \qquad It_k (\xi, i, \delta)$$

shall mean that modification of these two programs in which the program $\{\xi : A\}$ is substituted for the order $\{m : A\}$. A simple instance of such a modification is set up in Table XIII below.

150. It will be seen that the It_k for $k = 1, 2, 3, 4$ represent the four possible types IP, FS, FP, IS, respectively. It_5 is a modification of It_3 so as to make $i_F = i_Z$. It_6 is a modification of It_5 to adapt it for initial use in an iteration from 1 to m. The programs It_4 and It_6 are not used below.

151. An additional iteration procedure, It_7, for descending iteration, is formulated in Chapter V, Part C.

152. The characteristics of these various types can be seen by examining their uses in the inverse interpolation. They occur there in subdivisions I B, II B, III B, V B, VII C, and they form parts of IV A and VI.

153. Let us consider first the divisional control iterations I B, II B, III B. Here the preliminary program is the A part of the same division, the working program is the next division, and the terminal program is the C part of the same division (except in I, where the O_1 is open). The first question is whether the iteration control should be initial or final. In this case there is no a priori ground for favoring one over the other. For the reason already stated it has been decided to study both possibilities. It turns out there is little to choose between them, although the use of final control has the advantage that there is less breaking of the subdivisions into detached parts. The initial method iterates from 0 to m - 1, the final from 0 to m.

154. Suppose now that these iteration controls are initial. Then in cases II and III the terminal output is to C. But C also receives from the simple subdivision D. It would, therefore, be advantageous to make the terminal output of B secondary, so that D can go into C in a primary. This makes It_1 the appropriate iteration control. This is reinforced by the fact that It_1 is one order shorter than It_4. We use, then, It_1 in all three cases I, II, and III. The subdivision B is then broken into two parts B_1 (orders 1 - 4 in Table VI) and B_2 (orders 5 - 7);

subdivision A goes out to B_1 on a primary, B_1 goes out to the next division on a primary and to C (or O_1) on a secondary, the C of the next lower division feeds into B_2 on a primary, and subdivision D goes into C also on a primary. The iteration is from i = 0 to i = m-1, so that there are ℓ (instead of ℓ +1) groups, r rounds in each group, etc.

155. Suppose, on the other hand, that these iteration controls are final. Then the output of A goes directly to the working program on a primary. The working output from B must then go there on a secondary or a Kc order. This indicates that It_2 is the natural choice, but since It_2 involves one less order than It_2, It_3 with such a Kc order in the place of Oy of Oy is a possibility. If It_2 is used B goes to C on a primary, hence D will have to go to C on a Kc order. The iteration would be from 0 to m. If It_3 were used, B would go to C on a secondary, and hence D could go to C on a primary; the iteration would be from 0 to m-1. We make the supposition here that the problem calls for calculation from 0 to m*, and that D is an unimportant excrescence not always needed. These considerations determine the choice of It_2. But it must be admitted that under other circumstances It_3 would be preferable. If It_3 were programmed with $\{A \leq 0\}$ as fundamental discrimination, It_3 would also iterate from 0 to m.

156. Consider now the iteration control V B. This includes the adjustment $\{0 : u\}$. Here the working program is IV A and V A. The initial value of k is set in II A, and the question of whether k is to be increased or not depends on many other decisions beside that in V B. Hence this iteration control is necessarily final. Further the adjustment $\{0 : u\}$ has to be made immediately after discrimination so that it is an advantage to have it final. Again, when this iteration terminates a calculation is ordered which will eventually lead to a printing (in IV D), in which k must have the value it had before V B was entered. In other words it is important to have $i_f = i_z$. The only control problem with these characteristics is It_5. Hence

$$V\,B = \{It_5\,(k - q, K)\} \rightarrow \{0 : u\}\ .$$

157. Next we treat the iteration control VII C. This must also be final. Further, the working output must go to V D before the calculation can be repeated, while the terminal output is a simple error signal. Hence the iteration should be primary. There is no necessity to have $i_z = i_f$, and it is no disadvantage to have the iteration end with = n - 1. Hence

$$VII\,B = It_3\,(n, u).$$

*Actually it would probably be preferable to iterate from 0 to m - 1; then the working program operates m times. The more difficult situation is deliberately chosen here because the interest is in the programming.

158. As for the iteration in IV A, the situation is fairly simple. The dominating consideration is that $F_{\mu,\kappa+\lambda,\lambda}$ must go into x_1 from $i=-p$ to $i=q$ both inclusive. This can be done by using It_2. The terminal output then goes to the next discrimination on a primary.

D. Synthesis of Inverse Interpolation

159. Since the time allotted to this investigation is not sufficient to set up the complete program for the inverse interpolation by the methods described here, a partial synthesis must suffice to illustrate the principles. To obtain such a partial synthesis we shall construct two schedules: (1) a synthesis in detail of subdivision V A, and (2) a synthesis of the entire computation regarding the subdivisions as unanalyzed units. The latter synthesis will be performed twice with It_1 and It_2 for the divisional iteration controls.

160. The first of these projects is carried out in Tables VII and VIII. Table VII starts with the basic programs and builds up the program (63); Table VIII gives the further construction of subdivision V A. The method of constructing these tables is explained in Reference (a), par. 93; here the scheme is varied so that the first two columns give the final result, and the intermediate work is shown to the right. In Table VIII, the Y_1 and Y_2 are unanalyzed; for the present these must be interpreted as additional outputs into which those problems can be substituted later. The subdivision will be taken up again in Chapter V, Section A.

161. In the second project, the phrase "taking the subdivisions as unanalyzed units" means taking them as if they were basic programs whose data are the references to the secondary outputs, if any. Where a subdivision has one or more primary outputs in its interior, the subdivision is broken into parts and each part is taken as such a unit. All outputs leading outside the division are treated as separate units also. Only the order program is taken into account by this procedure. We start with programs for the separate divisions. These are given in Tables IX and XI. In these tables the primary outputs are indicated by giving in parentheses the input which is to be substituted there; in the case of secondary outputs, which are all put at the end, square brackets are used in the same way. Here O_1, O_2, O_3, O_4 are the outputs of the composite programs as a whole. The latter are constructed in Tables X and XII by the same technique as before. Horizontal lines here indicate that the previous item has no primary exit.

162. Flow charts for the two composite programs are given in Charts B and C. In these charts dashed lines indicate primary outputs, solid lines secondary outputs from a Kh order, lines with crosses secondary outputs from a Ke order, while the lines with circles indicate remote

control as discussed in Chapter V. These charts, along with Chart A, have illustrative value, but they were not used either in the planning or the construction of the program. The remote control in I A necessary to set up the entire scheme is not shown on these charts; for this, see Chapter V, Section E.

Table VII

Construction of $\{y \leqq x < z\}^2$

A Basic Programs

	$\{x : A\}$	$\{A-x : A\}$	$\{A-1 : A\}$	$\{A < 0\}$
1	+c 3	-h 3	- h U	Kh 3
2	0	0	0	0
3	x	x		0

B Construction of $\{x < y\}$ and $\{x \leqq y\}$.

	$\{x < y\}$	$\{x : A\}$	$\{A-y : A\}$	$\{A-1 : A\}$	$\{A < 0\}$ Remarks
1	+c 6	1			
2	-h 7	(2)	1		
3	Kh 5		(2)		1
4	0				2 0_1
5	0				3 0_2
6	x	3			
7	'y		3.		
	$\{x \leqq y\}$				
1	+c 7	1			
2	-h 8	(2)	1		
3	-h U		(2)	1	
4	Kh 6			(2)	1

Table VII (Cont'd.)

	$\{x \leqq y\}$	$\{x : A\}$	$\{A-y : A\}$	$\{A-1 : A\}$	$\{A < 0\}$	Remarks
5	0				2	0_1
6	0				3	0_2
7	x	3				
8	y		3			

C Final Construction

		$\{x < y\}$	$\{z \leqq x\}$			
1	+c 11	1	1			
2	-h 12	2				
3	Kh 5	3				
4	0	4				0_1
5	+c 13	(5)	1			
6	-h 11		2			
7	-h U		3			
8	Kh 10		4			
9	0		5			0_3
10	0		6			0_2
11	x	6	8			
12	y	7				
13	z		7			

Table VIII

Construction of State Bracket Test

			Y_1	$\{x_0 \leq a < x_1\}$	$\{a < x_{-p}\}$	Y_2	$\{x_1 < x_2\}$	Remarks
1	Y_1		1					
2	+c	19	(2)	1				
3	-h	21		2				
4	Kh	10		3				
5	+c	19		(4)	1			
6	-h	20			2			
7	Kh	9			3			0_3
8	0				4			
9	Y_2				(5)	1		
10	+c	22		5		(2)		
11	-h	19		6				
12	-h	U		7				
13	Kh	18		8				
14	+c	22		(9)			1	
15	-h	23					2	
16	Kh	18					3	
17	0						4	0_2
18	0			10			(5)	0_1

Table VIII (Cont'd.)

		Y_1	$\{x_0 \leq a < x_1\}$	$\{a < x_{-p}\}$	Y_2	$\{x_1 < x_2\}$	Remarks
19	a		11	6			
20	x_{-p}			7			
21	x_o		12				
22	x_1		13			6	
23	x_2					7	

Table IX

Division Programs Using IT_1

Item No.	I	II	III	IV	V	VI	VII	
1	A	A	A	A 6	A_1 3	A	A	
2	B_1 5	B_1 6	B_1 6	(VIA)	(IIID)	B	B 4,9	
3	(IIA)	(IIIA)	(IVA)	B 7	A_2 6	(IVB)	(IVD)	
4	B_2 2	B_2 2	B_2	C	B 6		C 10	
5	O_1	D	D	(IIIB_2)	[IVA]		D	
6		C	C	[VA]	[VIA]		E 8,11	
7		(IB_2)	(IIB_2)	[VIIA]			(IID)	
8				D 4			[VIB]	
9							[O_2]	
10							[O_3]	
11							[O_4]	

Table X

Composite Program Using It_1

	Item Data	I	II	III	IV	V	VI	VII
1	I A	1						
2	I B_1 32	2						
3	II A	(3)	1					
4	II B_1 27		2					
5	III A		(3)	1				
6	III B_1 15			2				
7	IV A 13			(3)	1	(5)		
8	VI A				(2)	(6)	1	
9	VI B						2	(8)
10	IV B 20				3		(3)	
11	IV C				4			
12	III B_2 6			4	(5)			
13	V A_1 17				(6)	1		
14	III D			5		(2)		
15	III C			6				
16	II B_2 4		4	(7)				
17	V A_2 8					3		
18	V B 8					4		
19	Kc 7					Kc to (5)		
20	VII A				(7)			1
21	VII B 23,29							2
22	IV D 11				8			(3)
23	VII C 30							4
24	VII D							5
25	VII E 9,31							6
26	II D		5					(7)
27	II C		6					
28	IB2 2	4	(7)					
29	O_2							9
30	O_3							10
31	O_4							11
32	O_1	5						

Table XI

Division Programs Using It_2

Item No.	I	II	III	IV	V	VI	VII
1	A .	A	A	A 6	A 3	A	A
2	(II A)	(III A)	(IV A)	(VI A)	(III D)	B	B 4,9
3	B 2	B 2	B 2	B 7	A_2 6	(IV B)	(IV D)
4	(O_1)	C	C	C	B 6		C 10
5		(I B)	(II B)	(III B)	[IV A]		D
6		D 4	D 4	[V A]	[VI A]		E 8,11
7				[VII A]			(II D)
8				D 4			[VI B]
9							[O_2]
10							[O_3]
11							[O_4]

Table XII

Composite Program Using It_2

	Item	Data	I	II	III	IV	V	VI	VII
1	I A		1						
2	II A		(2)	1					
3	III A			(2)	1				
4	IV A	15			(2)	1	(5)		
5	VI A					(2)	(6)	1	
6	VI B							2	(8)
7	IV B	20				3		(3)	
8	IV C					4			
9	III B	4			3	(5)			
10	III C				4				
11	II B	3		3	(5)				
12	II C			4					
13	I B	2	3	(5)					
14	Φ_1		4						
15	V A_1	17				(6)	1		
116	III D	10			6		(2)		
17	V A_2	5					3		
18	V B	5					4		
19	Kc	4					Kc to (5)		
20	VII A					(7)			1
21	VII B	23,27							2
22	IV D	8				8			(3)
23	VII C	28							4
24	VII D								5
25	VII E	6,29							6
26	II D	12		6					(7)
27	0 2								9
28	0 3								10
29	0 4								11
30									

CHAPTER V

Secondary Programs

163. In this chapter we consider programs which are secondary in the sense of Reference (a), par. 31. As before we consider only certain special cases.

A. Remote Control of an Output

164. The first case to be considered will be the situation illustrated by the control of the secondary output of the approximation test in case we allow an irregular calculation for $x_{-p} \leqq a < x_0$. In the normal situation the output O_4 of the inversion control leads to an error signal (output O_2 of the composite program), since it indicates failure of the monotone convergence. But if a calculation is ordered for $x_{-p} \leqq a < x_0$, the situation described by III - O_4 is not pathological. It is, therefore, necessary to adjust the discrimination order leading to VII - O_4 so that in this abnormal case the datum of this order is the start of VII C, and it must be provided that the situation is reset to normal before another calculation is ordered. The programs Y_1 and Y_2 in Chapter V, Section B were intended for this purpose. We consider here how they shall be constructed.

165. Let x be the location number of the relevant discrimination order in VII B. Let y_1 and y_2 be the location numbers of O_2 (VII - O_4) and of the start of VII C, respectively. If y_2 is in the accumulator, the order

Sd x

will put y_2 in the datum of the order at x. We can get y_2 in the accumulator if we have the number y_2 stored in the datum of an order (perhaps a storage order) located in location numbered Z_2. Then Y_2 could be a program consisting of the two orders

+c	Z_2
Sd	x

with an analogous arrangement for Y_1. We note that y_2 is available in the datum of the other discrimination order in VII B, but y_1 requires a special storage order. Thus the Y_1 and Y_2 require together five orders, and it is an accident of this particular problem that a sixth order is not required.

166. The necessity of storing y_1 and y_2 in any such case can be avoided if we use the order Gc mentioned in par. 77. Then the Y_2 can be simply

Gc	y_2
Sd	x

giving four orders for Y_1 and Y_2 together.

167. In the foregoing, Y_1 and Y_2 have been thought of as constitutents of the final composite program. But our object here is to formulate a simple program such that when combined with the other subprograms the Y_1 and Y_2 above described will arise automatically. Consider the following program:

1	Gc	5
2	Sd	4
3	0	
4	0	
5	0	

Here, order No. 3 is the output of the program in the usual sense, while 4 and 5 are additional secondary outputs.

168. The use of this program in composition is illustrated in the synthesis of the State Determination program in Table XII. In this table, and the others which follow, the constituent programs, or indications of them, are written at the top, then the place where the orders enter into the composite program is indicated below. The order of writing these constituent programs is irrelevant, and it is convenient to write the characteristic kernel, so to speak, of the program first even if it is quite simple, then to write those which amplify or modify it afterwards.

Table XIII

State Determination

		Table VIII	Y_1	Y_2	$\{It_5(A,K)\}$	$\{A-k+q:A\}$	$\{O:u\}$	Remarks
1			Ge 5	Ge 5	+e 10	-h 4	e Z	
2			Sd 4	Sd 4	-h 9	h 5	S 4	
3			O_1	O_1	Kh 8	O_1	O_1	
4			O_2	O_2	+e 10	k	u	
5			O_3	O_3	+h U	q		
6					S 10			
7					O_1			
8					O_2			
9					k			
10					k			
1	Ge 31	(1)	1					
2	Sd 30		2					
3	+e 36	2	(3)					
4	-h 39	3						
5	Kh 12	4						
6	+e 36	5						
7	-h 38	6						
8	Kh 10	7						
9	O_3	8						
10	Ge 32	(9)		1				to III 1
11	Sd 30			2				
12	+e 40	10		(3)				
13	-h 36	11						
14	-h U	12						
15	Kh 29	13						
16	+e 40	14						
17	-h 41	15						
18	Kh 29	16						
19	+e 35	(17)			1			
20	-h 33				(2)	1		
21	+h 34					2		
22	Kh 29				3	(3)		
23	e 35				4			
24	+h U				5			
25	S 35				6			

Table XIII (Cont'd.)

		Table VIII	Y_1	Y_2	{It_5(A,K)	{A-k+q:A}	{O: u}	Remarks
26	+c Z				(7)		1	
27	S 37						2	
28	O_2						3	to IV A
29	O_1	18			8			to VI A
30	O_4		4	4				to exit of VII B
31	O_5		5					to VII O_4
32	O_6			5				to VII C
33	k				9	4		
34	q					5		
35	K				10			
36	a	19					4	
37	u							
38	x_{-p}	20						
39	x_o	21						
40	x_1	22						
41	x_2	23						

169. In the further synthesis of the inverse interpolation problem, the outputs O_4, O_5, O_6 have to be identified with the first discrimination order in VII B, the error signal output O_2, and the output to VII B, respectively. It is evident that these are combinations which do not, strictly speaking, come under the theory developed in Reference (a), Chapter 4. In the first place they are not nondatum locations, but are actually nonexit locations. In the second place the locata being substituted are likely to occur in such a variety of locations in the interiors of programs that it would hardly be practical to specify them as inputs. This would make our notation break down. We shall not, however, pause to consider how the preceding theory can be generalized. It is evident that these combinations can be handled by the present technique, and that, at least in the cases considered here, they will do what they are supposed to do.*

170. The order Gc is not, of course, necessary for carrying out the above procedure. In fact, the program of par. 167 could be replaced by the following:

1.	~~Gc~~	5	
2.	Sd	4	
3.	O_1		
4.	O_2		(where substitution is made)
5.	0	6	(storage order)
6.	-	-	

Here location 6 is to be identified, in due course, with the locatum whose location number is to be substituted. A similar remark could be made if the order Gh were used. We shall use these Gc and Gh programs below with the understanding that, if desired, they can be dispensed with.

171. A situation which could also have been handled by this procedure of remote control arises at the end of a calculation. There the procedure is different in the cases $\mu = 0$ and $\mu > 0$. Since these two cases have been previously discriminated (in IV A), it would be possible for the first discrimination to set the output of VI B by remote control. But this would take more orders than to repeat the discrimination. In the case of pars. 164-169, however, the repetition of the discrimination would take many more orders, because x_0 is not immediately available.

---

*The substitution of {A - k+q : A} for {A - k : A} is also irregular but probably will not cause any difficulty.

172. Another case where remote control would be useful would be in connection with printing certain results in H. This occurs in II C, III C, and IV D. It would be possible to provide a program for reading from a certain auxiliary location into H, and to control the output by remote control. This possibility is considered in Section B below, but is not studied in detail.

B Read-Out and Tabulation Programs

173. We now consider programs involving reading out of tables and the like. A typical program of this sort is that for Subdivision IV A. We shall find such cases can be handled by the use of the order Gh. Thus consider the program

$$\begin{array}{lll} 1. & \text{Gh} & 5 \\ 2. & \text{Sd} & 4 \\ 3. & O_1 & \\ 4. & O_2 & \\ 5. & - & \end{array} \qquad (73)$$

where the dash at word 5 indicates that it is irrevalent what its contents are. Suppose we start this program with a quantity ξ in A. If we substitute for O_1 a program X and identify O_2 with an order at location α of X, so that 5 takes the location e in the resulting program, then the effect will be to replace the datum in that order by $e + \xi$.

174. In the cases where only quantities are involved the following notation is convenient. If α is a location number we shall denote by

$$L(\alpha)$$

the word in the location whose number is α . This notation can presumably be conceived, like that of locatum, in such a way as to be invariant of certain compositions - e.g., by referring it to a reference program.

175. Now let X be a program containing a locatum x. Then we shall designate by the notations

$$\begin{array}{l} X\{L(e+\xi)\} \\ \{\eta = L(e+\xi)\} \rightarrow X(\eta) \end{array} \qquad (74)$$

a program which has the effect of X with $L\,(e \dashv \xi)$ substituted for x. If X refers to x in only one order, and that is in position α, then the program (74) can be realized by substituting X in (73) as above described. In other cases this will not be so, so that the prefix $\{\varkappa = L\,(e \dashv \xi)\}$ is not defined apart from some program X to which it applies. When X is given, there is a program, which we can call $\{\varkappa = L\,(e \dashv \xi)\}$, such that we get (74) by substituting X in it. Some examples of such programs are given below. In such a case we shall call X the root program and $\{\varkappa = L\,(e \dashv \xi)\}$ the substitution program. We note, incidentally, that

$$X\,(L\,(e \dashv \xi)) = \{\xi \cdot A\} \rightarrow X\,(L\,(e \dashv A)).$$

Table XIV

Program for Phase Setup (IV A)

$\{-\phi : i\} \rightarrow \{\alpha : A\} \rightarrow \{x = L(\alpha)\} \rightarrow \{\beta : A\} \rightarrow \{y = L(\beta) \rightarrow \{x : y\} \rightarrow \{It_2(y, i)\} \begin{cases} \rightarrow \{\mu > 0\} \begin{cases} \rightarrow O_1 \\ \rightarrow O_2 \end{cases} \\ \rightarrow I_2^* \end{cases}$

		$\{x : y\}$	$\{x = L(\alpha)\}$	$\{y = L(\beta)\}$	$\{\alpha : A\}$	$\{\beta : A\}$	$\{-\phi : i\}$	$\{It_2(q,i)\}$	$\{\mu > 0\}$
1		+c 4	Gh 5	Gh 5	+c 14	+c 4	-c 4	+c 10	-c 5
2		S 5	Sd 4	Sd 4	+h U	+h 5	S 5	+h U	Kh 4
3		O_1	O_1	0	R A	0	O_1	S 10	O_1
4		x	O_2	O_2	X 17	p	p	+c 9	O_2
5		y	x	x	+h 16	i	i	-h 10	μ
6					+h 19			Kh 8	
7					S 20			O_1	
8					+c 15			O_2	
9					+h U			q	
10					R A			i	
11					X 20				
12					+h 18				
13					O_1				
14					k				
15					m				
16					κ				
17					λ				
18					μ				
19					i				
20					w				
1	-c 35						1		
2	S 40						2		
3	+c 33				1		(3)	(8)	
4	+h U				2				
5	R A				3				

NOLM 10337

81

Table XIV (Cont'd.)

			$\{x : y\}$	$\{x = L(\alpha)\}$	$\{y = L(\beta)\}$	$\{\alpha : A\}$	$\{\beta : A\}$	$\{-\mu : i\}$	$\{It_2(q,i)\}$	$\{\mu > 0\}$
6	X	38				4				
7	+h	37				5				
8	+h	40				6				
9	S	41				7				
10	+c	34				8				
11	+h	U				9				
12	R	A				10				
13	X	41				11				
14	+h	39				12				
15	Gh	43		1		(13)				
16	Sd	21		2						
17	+c	35		(3)			1			
18	+h	40					2			
19	Gh	42			1		(3)			
20	Sd	22			2					
21	+c	-	1	(4)	(3)					
22	S	-	2		(4)					
23	+c	40	(3)						1	
24	+h	U							2	
25	S	40							3	
26	+c	36							4	
27	-h	40							5	
28	Kh	3							6	
29	-c	39							(7)	1
30	Kh	32								2
31	O_2									3
32	O_1									4

Table XIV (Cont'd.)

		$\{x : y\}$	$\{x=L(\alpha)\}$	$\{y=L(\beta)\}$	$\{\alpha : A\}$	$\{\beta : A\}$	$\{\varphi : i\}$	$\{It_2(q,i)\}$	$\{\mu>0\}$
33	k				14				
34	m				15				
35	p						4 4		
36	q							9	
37	k				16				
38					17				
39					18				
40	i				19		5 5	10	5
41	w				20				
42	L(e)	(4)		5					
43	L(f)	(5)	5						

NOLM 10337

176. The main part of the program IV A is a program

$$\{L(f+\alpha) : L(e+\beta)\}$$

where

$$\alpha = \mu + (m+1)[\kappa + i + (k+1)\lambda]$$

$$\beta = i + \rho \ .$$

This may be formulated as

$$\{\alpha : A\} \to \{\varkappa = L(A+f)\} \to \{\beta : A\} \to \{\eta = L(A+e)\} \to \{\varkappa : \eta\}$$

Thus - if we extend the definitions just made in an obvious manner to two substitutions - the root program is $\{\varkappa \to \eta\}$. In formulating this and other similar programs it seems to be convenient to formulate the root program first, then the substitution program or programs when the ξ is in A, then the programs to put the right quantities in A. After that we can consider iteration processes, if iteration is necessary - including the preliminary program. Finally we can tack on any extras which may be needed.

177. The complete program for subdivision IV A is put together in this way in Table XIV.

178. Evidently the printing programs in IV C can be set up similarly. As for the printings in II C, III C, and IV D, which go into H, the simplest arrangement is have them go into H as they come; i.e., we can have a counter which is increased by a unit every time a word is sent into H, and this can be taken as the ξ to send the next word into the next position. Perhaps this is not a realistic arrangement, but the matter will not be gone into further.

179. The word (5) in (73) is purely a dummy. In the building up of the composite program it is presumed that it will be identified with a word occupying the corresponding location. Thus word 42 in Table XIV will eventually be identified with X_{-p}, word 43 with F_{ooo}.

180. Just as in Section A these combinations can be brought about, without using the orders G, by using storage orders.

181. Whether there is any utility in extending these considerations to cases where the words involved are not quantities has not been investigated.

C Programs Related to the Calculation

182. The programs for carrying out the calculations are similar in character to those in Section B. They are necessarily secondary programs if p and q are unspecified.

Table XV

Difference Formation

$\{1:i\} \rightarrow \{s:j\} \rightarrow \{j:A\} \rightarrow \{x = L(e+j)\} \rightarrow \{y = L(e+j-1)\} \rightarrow \{x-y:x\} \rightarrow It_1(i,j) \begin{cases} \rightarrow It_2(s,i) \begin{cases} \rightarrow O_1 \\ \rightarrow I_2 \end{cases} \\ \rightarrow I_3 \end{cases}$

	$\{x-y:x\}$	Subst.	$\{j:A\}$	$\{1:i\}$	$\{s:j\}$	$\{It_2(s,i)\}$	$\{It_7(i,j)\}$
1	+c 5	Ch 10	+c 3	+c U	+c 4	-c 10	+c 9
2	-h 6	Sd 7	O_1	S 4	S 5	+h U	-h U
3	S 5	Sd 8	j	0	0	S 10	S 9
4	0	-h U		i	s	+c 9	-h 8
5	x	Sd 9			j	h 10	Kh 7
6	y	O_1				Kh 8	O_1
7		O_2				O_1	O_2
8		O_3				O_2	i
9		O_4				s	j
10		x				i	
1 +c U				1			
2 S 27				2			
3 +c 26				(3)	1	(8)	
4 S 28					2		
5 +c 28			1		(3)		(7)

Table XV (Cont'd.)

			{x-y : x}	Subst.	{j : A}	{l : i}	{s : j}	$\{It_2 (s,i)\}$	$\{It_7 (i,j)\}$
6	Gh	29		1	(2)				
7	Sd	11		2					
8	Sd	13		3					
9	-h	U		4					
10	Sd	12		5					
11	+c	-	1	(7)(6)					
12	-h	-	2	(9)					
13	S	-	3	(8)					
14	+c	28	(4)						1
15	-h	U							2
16	S	28							3
17	-h	27							4
18	Kh	5							5
19	+c	27						1	(6)
20	+h	U						2	
21	S	27						3	
22	+c	26						4	
23	-h	27						5	
24	Kh	3						6	
25	O_1							7	

Table XV (Cont'd.)

		{x-y : x}	Subst.	{j ↑ A}	{1 : i}	{s : j}	{It$_2$ (s,i)}	{It$_7$ (i,j)}
26	s					4	9	
27	i				4		10	8
28	j			3		5		9
29	x	(5)						
30	-	(6)						

Table XV (Cont'd.)

Table XVI

Division by Factorials

		$\{\frac{\alpha}{\alpha\pi} : x\}$	Substitution Program	$\{i : A\}$	$\{\omega : \nu\}$	$\{I : i, v\}$	$\{It_2(s,i)\}$
1		+c 7	Gh 7	+c 3	R 5	+c U	+c 10
2		4 6	Sd 5	0_1	X 6	S 5	+h U
3		+c R	Sd 6	i	S 5	S 6	S 10
4		S 7	0_1		0_1	0	+c 9
5		0_1	0_2		v	v	-h 10
6		v	0_3		i	i	Kh 8
7		x	x				0_1
8							0_2
9							s
10							i
1	+c U					1	
2	S 24					2	
3	S 23					3	(3)
4	R 24				1	(4)	(8)
5	X 23				2		
6	S 24				3		
7	+c 23			1	(4)		
8	Gh 25		1	(2)			
9	Sd 11		2				
10	Sd 14		3				
11	+c --	1	(5)(4)				
12	s 24	2					
13	+c R	3					
14	S --	4	(6)				1
15	+c 23	(5)					
16	+h U						2
17	S 23						3
18	+c 22						4
19	-h 23						5
20	Kh 4						6
21	0_1						7
22	s						9
23	i			3	6	6	10
24	v	6			5	5	
25	x	(7)	7				
26							
27							
28							
29							
30							

Table XVII

Centered Newton Formula

		{x÷vw:w}	Subst.	{i : A}	{u+p=i : ω}	{s : i}	{o : v}	{It$_7$(o,i)}
1		R 6	Gh 5	+c 3	+c 8	+c 4	+c Z	+c 7
2		X 7	Sd 4	0_1	+h 6	S	S 4	-h U
3		+h 8	0_1	i	-h 7	0	0	S 7
4		S 6	0_2		S 9	s	v	Kh 6
5		0	x		0_1	i		0_1
6		v			p			0_2
7		w			i			i
8		x			u			
9					w			
1	+c Z						1	
2	S 25					1	2	
3	+c 22					2	(3)	
4	S 23				1	(3)		(6)
5	+c 24							
6	+h 21				2			
7	-h 23				3			
8	S 26			1	4			
9	+c 23		1	(2)	(5)			
10	Gh 27							
11	Sd 14		2					
12	R 25	1	(3)					
13	X 26	2						
14	+h -	3	(4)					
15	S 25	4						
16	+c 23	(5)						1
17	-h U							2
18	S 23							3
19	Kh 5							4
20	0_1							5
21	p				6	4		
22	s			3	7	5		7
23	i				8			
24	u						4	
25	v	6						
26	w	7			9			
27	x=L(o)	(8)	5					
28								
29								
30								

183. If we suppose the calculation is carried out by the "centered Newton formula" used in Reference (b), there will be three steps in the calculation, viz.

1) formation of differences,

2) division by factorials, and

3) final calculation.

The first two of these will constitute Subdivision VI A;* the third is Subdivision VI B. The three stages are set up below in Tables XV - XVII. The three root programs are, respectively,

$$\{\nu - \tfrac{\nu}{2} : \nu\}$$
$$\{\nu/\omega \cdot \nu\}$$
$$\{\nu + \nu\omega : \nu\} .$$

184. In connection with differences we have a double iteration. It is convenient to make the inside iteration descending, i.e., to start with $J = S$ and descend to and including $j = i$. For this purpose a new iteration-control program It7 has been constructed. It appears in Table XV.

185. It will be noted that these tables are based on the assumption that there are $s+1 = p+q+1$ coefficients in our interpolation formula. The tables are believed to need no further comment.

D Preparatory Routines

186. In their report, Reference (d), Goldstine and von Neumann code what they call a preparatory routine. The simpler of these, their Problem 16, is a device for picking out a certain segment of the memory, and increasing all the datum numbers of the orders in that segment by a fixed amount, provided those numbers originally lay between certain bounds. It is assumed, apparently, that all the words in the segment are orders. The more general routine, Problem 17, is an iteration of this over several intervals.

187. The coding in Reference (d) is complicated by details of the position of the unit point. In the idealized situation we have postulated here, these considerations are irrelevant. Without them the program is set up, using the present technique, in Table XVIII. Unfortunately time is not available to consider the more complex problem.

---

*VI A should also set $\nu = 0$, so that the U locatum can be used as an auxiliary up to that point.

Table XVIII

Preparatory Routine

		$\{A+\delta:\alpha(x)\}$	Dc	$\{b\leqq A<b+\beta\}$	Substitution Program	$\{It_1(h,i)\}$	$\{0:A,w\}$	$\{i:A\}$
1		+h 4	+c 6	-h 10	Gh 7	S 10	+c Z	+c 3
2		Sd 5	Sd 5	-h 11	Sd 5	-h 9	S 4	01
3		0_1	+c 5	Kh 9	Sd 6	Kh 8	0_1	1
4		8	0_1	+h 11	0_1	0_1	w	
5		x	w	Kh 7	0_2	+c 10		
6			x	Kc 9	0_3	+h		
7				+h 10	e	Kc 1		
8				0_1		0_2		
9				0_2		h		
10				b		i		
11				β				
1	+c Z						1	
2	S 31						2	
3	S 28					1	(3)	
4	-h 27					2		
5	Kh 25					3		

NOTT 10337

76

Table XVIII (Cont'd.)

			$\{A+\delta : \alpha(x)\}$	Dc	$\{b \leq A < b+\beta\}$	Substitution Program	$\{It_1(h,i)\}$	$\{0 : A, \omega\}$	$\{i : A\}$
							(4)		1
6	+c	28				1			(2)
7	Gh	26				2			
8	Sd	10				3			
9	Sd	21				(5)(4)			
10	+c	-		1					
11	Sd	31		2					
12	+c	31		3					
13	-h	29		(4)	1				
14	-h	30			2				
15	Kh	22			3				
16	+h	30			4				
17	Kh	19			5				
18	Kc	22			6				
19	+h	29			7				
20	+h	32	1		(8)				
21	Sd	-	2			(6)			
22	+c	28	(3)		(9)		5		

Table XVIII (Cont'd)

		$\{A+\delta : \alpha(x)\}$	Dc	$\{b \leqq A < b+\beta\}$	Substitution Program	$\{It_1(h,i)\}$	$\{0 : A, \omega\}$	$\{i : A\}$
23	+h U					6		
24	Kc 3					7		
25	O_1					8		
26	L(e)	(5)	(6)		7			
27	h					9		
28	i					10		3
29	b			10				
30	β			11				
31	ω		5				4	
32	δ	4						

NOLM 10337

93

Table XVIII (Cont'd.)

188. In Table XVIII the column "Dc" is a program for extracting the datum number - or part of the word corresponding to the set of digits which constitute the datum number in an order - from a word in the memory and storing it in A. The fourth column of the table is a test for $\alpha \leqq A < \alpha + \beta$ without requiring the word in A to be put in the memory.

E Main Control

189. A few remarks will now be made about the assembly of the complete program for inverse interpolation.

190. It is evident that the divisional programs contain explicit reference to the following locata in parts E, F, G, H of the arrangement of Chapter I, Section D:

in E x_{-p}, x_0, x_1, x_2,

in F F_{ooo},

in G G_{ooo}, and

in H H_o.

The last three of these occur only in the data of Gh orders.

191. Suppose now we assemble the composite program assigning to these locata the numbers in the second column of the following table

x_{-p}	e	e
x_o	e + 1	e + p
x_1	e + 2	e + p + 1
x_2	e + 3	e + p + 2
F_{ooo}	e + 4	f
G_{ooo}	e + 5	g
H_o	e + 6	h

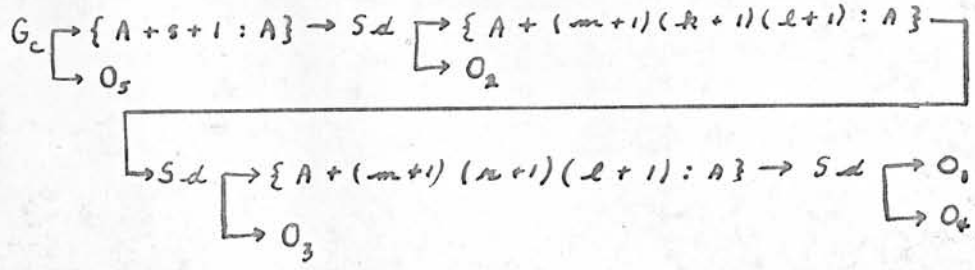
The true location numbers of these locata are those in the third column. (Note: f, g, h are defined by (21).)

192. Now we can evidently go from the second column to the third by a preparatory routine of the sort just described. However, although that has some automatic features to recommend it, it seems more

complicated than is really necessary. A substitution program could be constructed along the lines of Sections B and C which would be simpler. This substitution program could be divided into two parts. A first part, at the beginning of V, would be:

1.	Gc	14	
2.	+h	15	
3.	Sd	10	
4.	+h	U	
5.	Sd	11	
6.	Sd	12	
7.	+h	U	
8.	Sd	13	
9.	O_1	to Table XIII	1
10.	O_2	to Table XIII	12
11.	O_3	to Table XIII	20
12.	O_4	to Table XIII	24
13.	O_5	to Table XIII	25
14.		(Table XIII	38)
15.	p		

where the outputs are connected to those in Table XIII as indicated. The other, in I A, would be as follows



Here O_2 is to be identified with 43 in Table XIV, O_3 and O_4 with similar places where L(g) and L(h) are stored in IV C and II D, III D, or IV D,

while 0_5 is to be identified with 38 in Table XIII (which is the same as 42 in Table XIV). The programs $\{A + (m+1)(k+1)(l+1) : A\}$ can be carried out as in Chapter III. That will require two auxiliary locations, which can be taken from those used for $\lambda, \nu, \mu, \rho, \gamma$ etc.

193. This matter cannot, however, be looked into further.

H. B. Curry

HBC:ces

APPENDIX

In the course of the preparation of this memorandum there were certain improvements and changes which were not thought of in time or are of too miscellaneous a nature to be incorporated in the text. These ideas are discussed briefly here.

1. The fundamental multiplication order was taken from Reference (c) in the form $\{xR : A\}$. But the discussion in Chapter III suggests that it would be advantageous to take this fundamental order in the form $\{xA : A\}$. This could be done by programming an initial $\{A : R\}$ as part of the built-in multiplication subroutine. The multiplication would take one time-unit of some sort longer. But since this time-unit is small in proportion to the time necessary for a multiplication, and since further memory-space is likely to be at a premium, the saving in memory-space will more than compensate for the increased time. The programming will also be simpler. There will then be no occasion to put words into R except through A, and since they cannot come out except that way either, it will be possible, in effect, to ignore the possibility $\lambda = R$ in the theory of Chapter III. With this change there will be a reduction from ten orders to eight in Table IV; the program (47) will have ten orders, while (49) and (50) will have the same number of orders, viz., nine. Thus the various ways of calculating the quantity (46) will hardly differ from one another, and experience indicates that it is frequently not worth while to investigate different ways of calculating a quantity. Likewise the program of Table XIV would be two orders shorter.

2. It would simplify the programming in Chapter III if similarly the quotient, after a division, were immediately transferred to A. But such an arrangement would run into the difficulty that the remainder of the division is normally in A, and that for some purposes it is necessary to conserve that remainder. Since division does not occur as frequently as multiplication, it is probably best to use the orders as they are. But there are other possibilities which might be considered. There could be two division orders, one as at present, the other with such a transfer into A. Another is to provide the a. u. with a second register, and to provide that the remainder be put in it automatically. This second register could be simply a position in the memory reserved for the purpose, and it could be used as auxiliary location, if necessary, for the reductions considered in Chapter II, Section D.

3. In connection with the discussion of design of the machine the most important considerations are efficient use of the memory and efficiency in programming. The size of the memory determines the kind of problems the machine can hanile. But given a certain memory capacity, the principal bottleneck for efficient performance is the preparation of problems for the machine. Consequently features of machine design which will cause an improvement in programming technique should be very seriously considered.

4. In multiplication, different parts of the answer appear in both A and R. In accordance with the idealizations made here this consideration has been ignored. But when the factor is taken into consideration there may be uses for two multiplication orders which will put either of the two parts of the answer into A.

5. In the Goldstine and von Neumann system orders are words of half length and are stored in the memory in pairs. But there are many quantities which are also suitable for storage in half-length words. It would seem likely that arrangements could be made to take care of such half-length words without an undue increase in complexity. There should, for example, be orders for interchanging the two halves of A and for clearing either half separately.* /

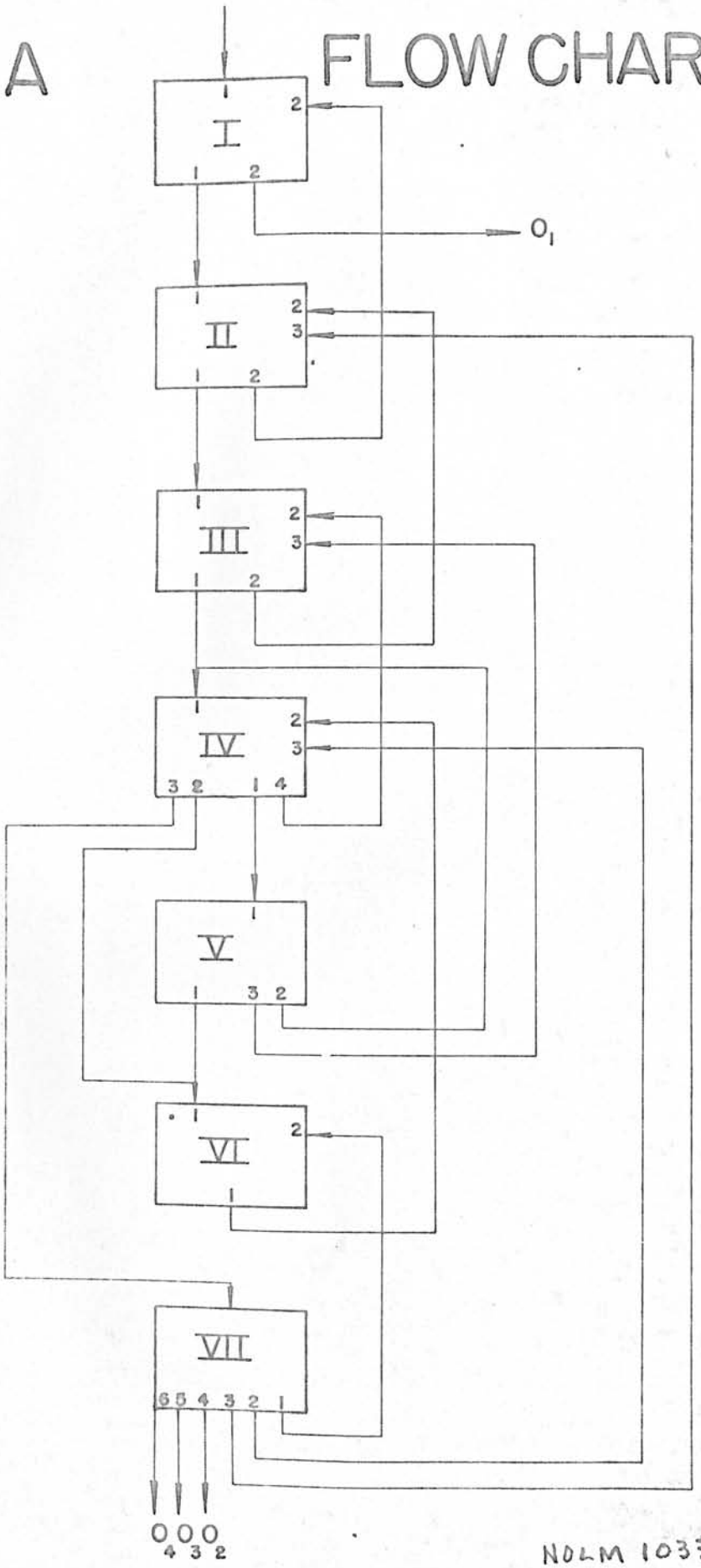
6. The fact that it takes three orders to replace i by i+1, and the fact that the half-length orders cause some complexity, is an argument for using full-length orders with multiple data. Such systems are used on some other machines. With 12-digit location numbers and 40-digit words there could be 15 orders with 3 data plus 2^{12} orders with 2 data. The principle of automatic shift of control to the next memory position could be used here also. Then orders with three data could be used for binary operations, two data being for the operands and the third for the result. Orders with two data could be used for shifts, substitutions, etc. Of course a theory of composition could be erected on such a basis. It is not clear whether the use of the memory would be more or less efficient. For although $\{x + y : z\}$ could be formed with one full-length word instead of three half-length words, $\{x + y + z + w : u\}$ would require three full-length words (if no auxiliary locata are used) in the multiple-address system, five half-length words in the single-address system. The inverse-interpolation problem is presumably not a good one for testing this out, because it contains too much logic and too little arithmetic.

7. The inconvenience that k is the same for all λ could be taken care by an auxiliary test as in (b). This would have to be in V a if it had to take place after a read out. Other complications along this line are possible.

---

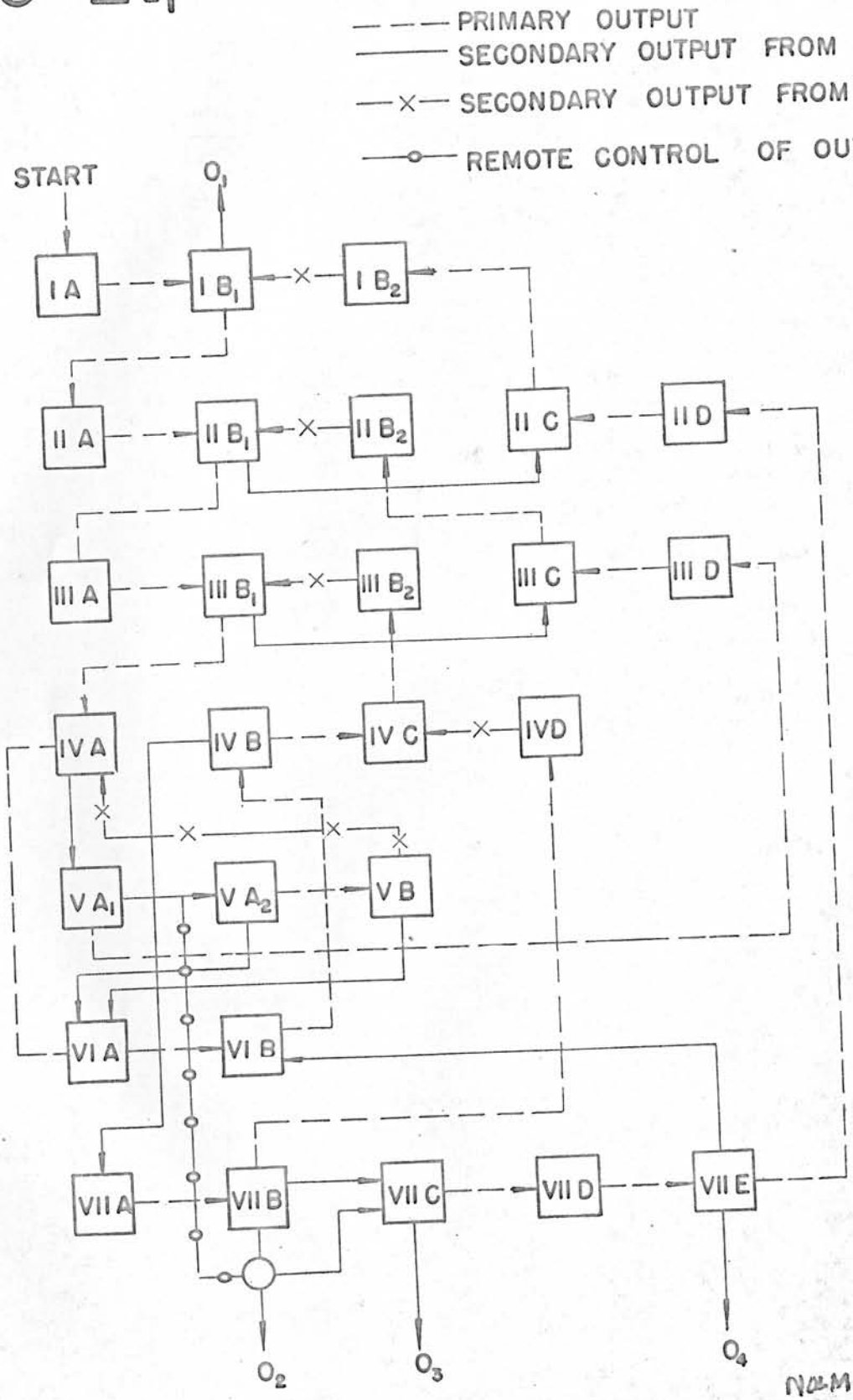
*In the revised scheme of basic orders for the Goldstine and von Neumann machine some of these possibilities are provided for.

CHART A FLOW CHART



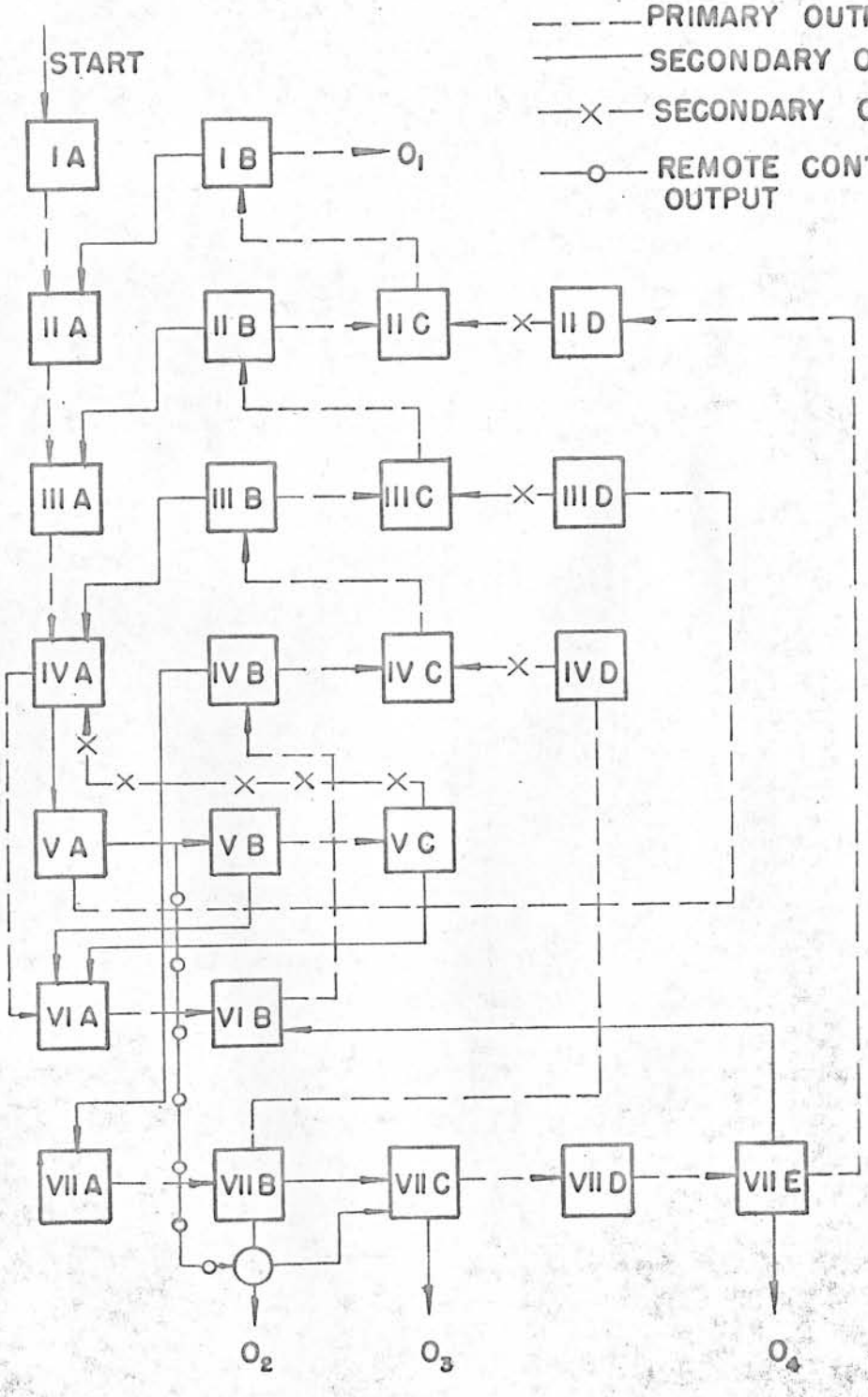
NOLM 10337

CHART B COMPOSITE PROGRAM USING It_1



NAM 10337

CHART C COMPOSITE PROGRAM USING It_2



NOLM 10337