THE EDUCATION OF A COMPUTER

Grace Murray Hopper
Remington Rand Corp.

While the materialization is new, the idea of mechanizing mathematical thinking is not new. Its lineage starts with the abacus and descends through Pascal,Leibnitz, and Babbage. More immediately, the ideas here presented originate from Professor Howard H. Aiken of Harvard University, Dr. John W. Mauchly of Eckert-Mauchly and Dr. M. V. Wilkes of the University of Cambridge. From Professor Aiken came, in 1946, the idea of a library of routines described in the Mark I manual, and the concepts embodied in the Mark III coding machine; from Dr. Mauchly, the basic principles of the "short-order code" and suggestions, criticisms, and untiring patience in listening to these present attempts; from Dr. Wilkes, the greatest help of all, a book on the subject. For those of their ideas which are included herein, I most earnestly express my debt and my appreciation.

Introduction

To start at the beginning, Fig. 1 represents the configuration of the elements required by an operation: input to the operations; controls, even if they be only start and stop; previously prepared tools supplied to the operation; and output of products, which may, in turn, become the input of another operation. This is the basic element of a production line; input of raw materials, controlled by human beings, possibly through instruments; supplied with machine tools; the operation produces an automobile, a rail, or a can of tomatoes.

The armed services, government, and industry are interested not only in creating new operations to produce new results, but also in increasing the efficiency of old operations. A very old operation, Fig. 2, is the solution of a mathematical problem. It fits the operational configuration: input of mathematical data; control by the mathematician; supplied with memory, formulas, tables, pencil, and paper; the brain carries on the arithmetic, and produces results.

It is the current aim to replace, as far as possible, the human brain by an electronic digital computer. That such computers themselves fit this configuration may be seen in Fig. 3. (With your permission, I shall use UNIVAC* as synonymous with electronic digital computer; primarily because I think that way, but also because it is convenient.)

Adding together the configurations of the human being and the electronic computer, Fig. 4 shows the solution of a problem in two levels of operation. The arithmetical chore has been removed from the mathematician, who has become a programmer, and this duty assigned to the UNIVAC. The programmer has been supplied with a "code" into which he translates his instructions to the computer. The "standard knowledge" designed into the UNIVAC by its engineers, consists of its elementary arithmetic and logic.

This situation remains static until the novelty of inventing programs wears off and degenerates into the dull labor of writing and checking programs. This duty now looms as an imposition on the human brain. Also, with the computer paid for, the cost of programming and the time consumed, comes to the notice of vice-presidents and project directors. Common sense dictates the insertion of a third level of operation, Fig. 5.

* Registered trade mark.

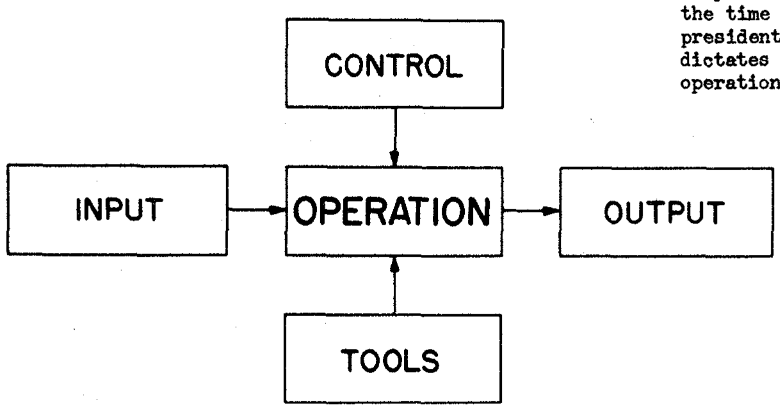

Fig. 1 - AN OPERATION

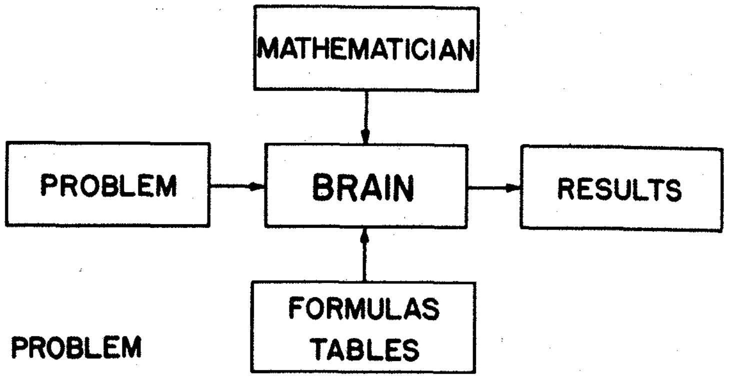

Fig. 2 - SOLUTION OF PROBLEM

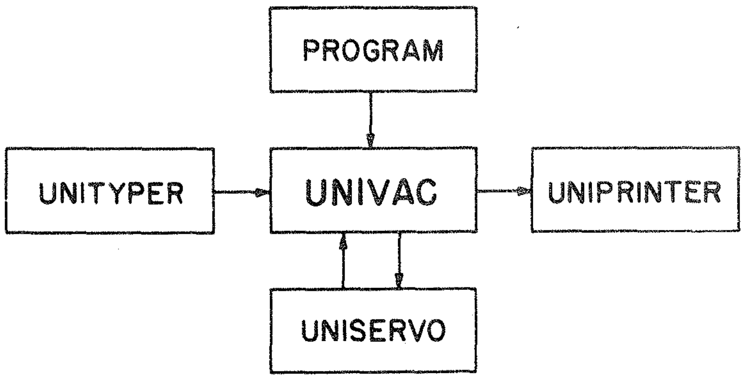

Fig. 3 - UNIVAC SYSTEM

The programmer may return to being a mathematician. He is supplied with a catalogue of subroutines. No longer does he need to have available formulas or tables of elementary functions. He does not even need to know the particular instruction code used by the computer. He needs only to be able to use the catalogue to supply information to the computer about his problem. The UNIVAC, on the basis of the information supplied by the mathematician, under the control of a "compiling routine of type A", using subroutines and its own instruction code, produces a program. This program, in turn directs the UNIVAC through the computation on the input data and the desired results are produced. A major reduction in time consumed and in sources of error has been made. If the library is well-stocked, programming has been reduced to a matter of hours, rather than weeks. The program is no longer subject either to errors of transcription or of untested routines.

Specifications for computer information, a catalogue, compiling routines, and subroutines will be given after adding another level to the block diagram. As Fig. 5 stands the mathematician must still perform all mathematical operations, relegating to the UNIVAC programming and computational operations. However, the computer information delivered by the mathematician no longer deals with numerical quantities as such. It treats of variables and constants in symbolic form together with operations upon them. The insertion of a fourth level of operation is now possible, Fig. 6. Suppose, for example, the mathematician wishes to evaluate a function and its first n derivatives. He sends the information defining the function itself to the UNIVAC. Under control of a "compiling routine of type B", in this case a differentiator, using task routines, the UNIVAC delivers the information necessary to program the computation of the function and its derivatives. From the formula for the function, the UNIVAC derives the formulas of the successive derivatives. This information processed under a compiling routine of Type A yields a program to direct the computation.

Expansion makes this procedure look, and seem, long and complicated. It is not. Reducing again to the two-component system, the mathematician and the computer, Fig. 7 presents a more accurate picture of the computing system.

Presuming that code, program, input data, and results are familiar terms, it remains to define and specify the forms of information and routines acceptable to this system. These include

catalogue,
computer information,
subroutine,
compiling routines, type A and B,
and task routines.

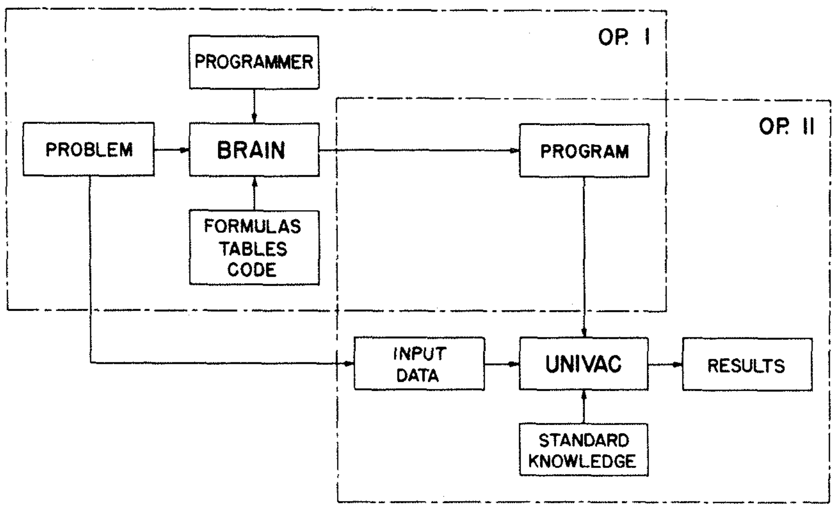

Fig. 4 - SOLUTION OF A PROBLEM

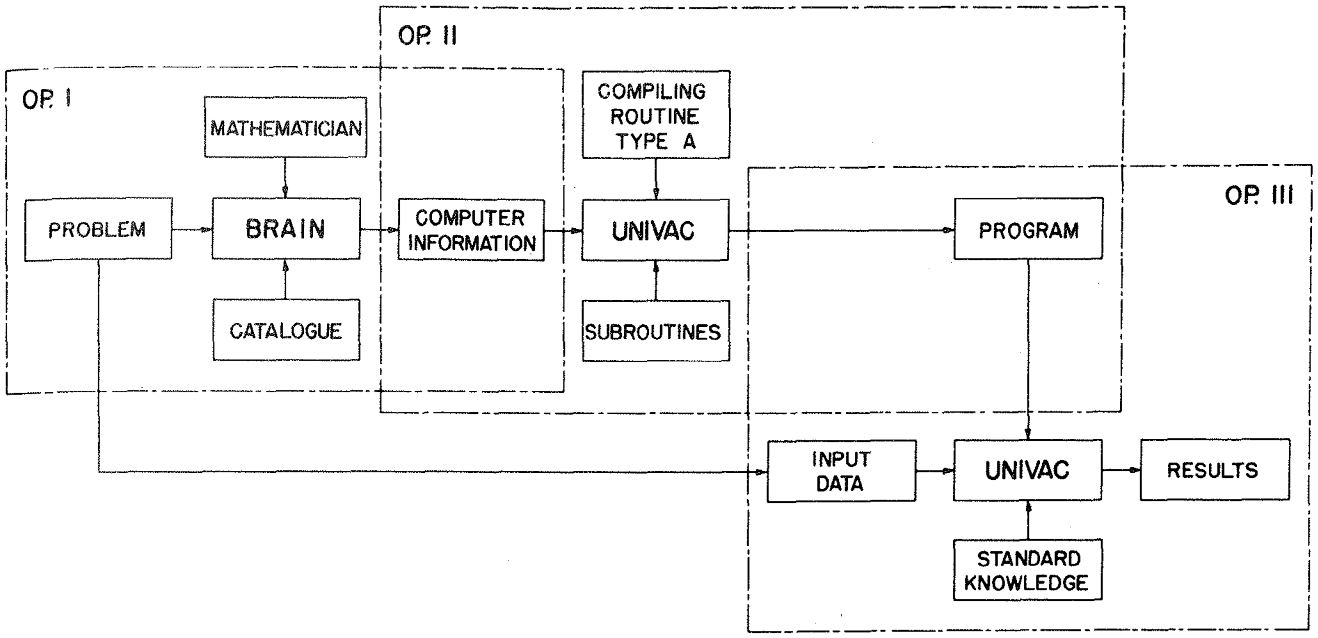

Fig. 5 - COMPILING ROUTINES AND SUBROUTINES

Catalogue and Computer Information

As soon as the purpose is stated to make use of subroutines, two methods arise. In one, the program refers to an immediately available subroutine, uses it, and continues computation. For a limited number of subroutines, this method is feasible and useful. Such a system has been developed under the nick-name of the "short-order code" by members of the staff of the Computational Analysis Laboratory.

The second method not only looks up the subroutine, but translates it, properly adjusted, into a program. Thus, the completed program may be run as a unit whenever desired, and may itself be placed in the library as a more advanced subroutine.

Each problem must be reduced to the level of the available subroutines. Suppose a simple problem, to compute

$$y = e^{-x^2} \sin cx,$$

using elementary subroutines. Each step of the formula falls into the operational pattern, Fig. 8 ; that is,

$$u = x^2$$

$$U = e^{-u}$$

$$v = cx$$

$$V = \sin v$$

$$y = UV.$$

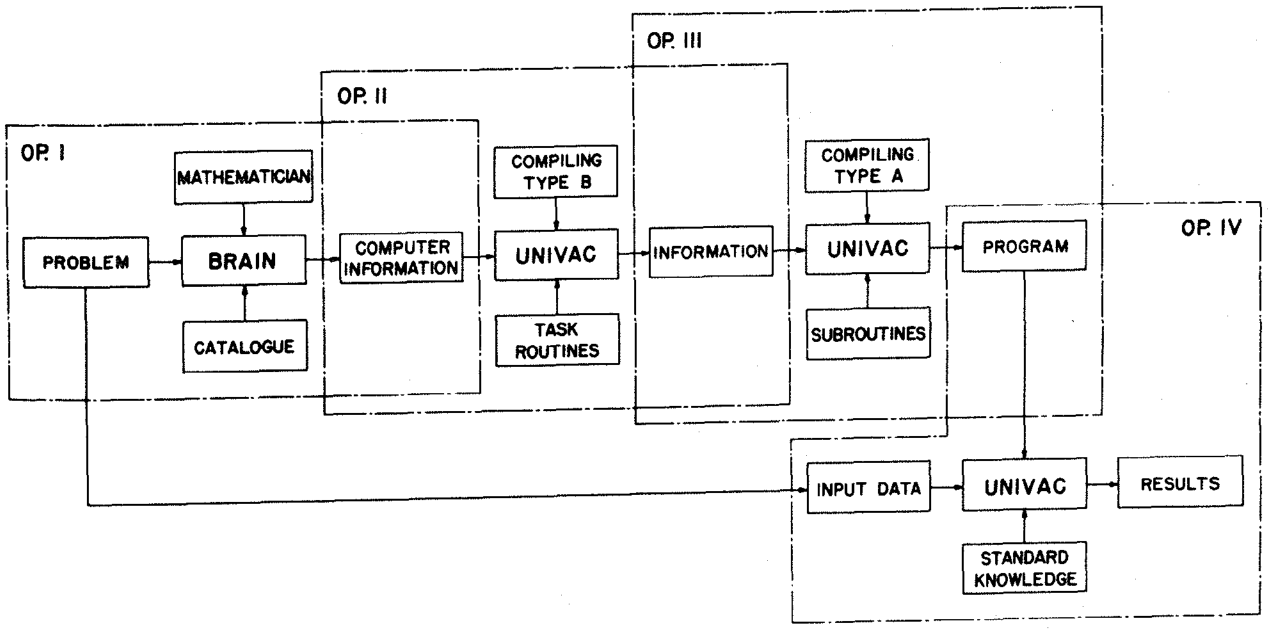

Fig. 6.- COMPILING TYPE B AND TASK ROUTINES

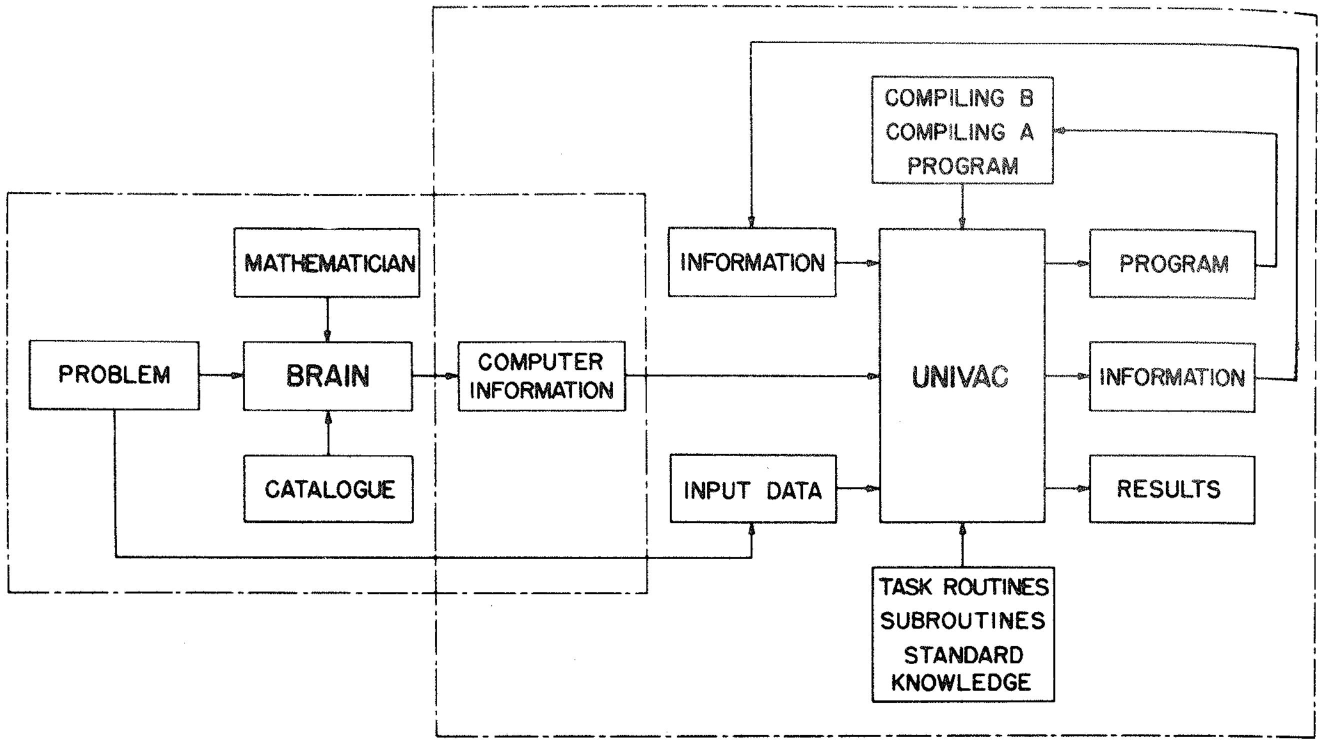

Fig. 7 - COMPUTING SYSTEM

As presented in Fig. 9, however, this information is not yet sufficiently standardized to be acceptable to a compiling routine. Several problems must be considered and procedures defined.

The operations are numbered in normal sequence and this number becomes part of the computer information. Thus when it is desired to change the normal sequence, the alternate destination is readily identified. The compiling routine translates these operation numbers into instructions in the coded program. Two fundamental situations arise, the alternate destination either precedes the operation under consideration or follows it, by-passing several intermediate operations. In both cases, it is necessary only to have the compiling routine remember where it has placed each subroutine or that a transfer of control to operation k has been indicated. In any event the mathematician need only state, "go to operation k", and the compiling routine does the rest.

The symbols to be used for the arguments and results, as well as for the operations, are of next concern. One mathematician might write

$$y = e^{-x^2} \sin cx$$

and another

$$u = e^{-v^2} \sin gv.$$

The obvious solution proves best. Make a list of arguments and results and number them. (This amounts to writing all constants and variables as x_i.) The order is immaterial, so that forgotten quantities can be added at the end.

1	x	x_1	6	ç	x_6
2	Δx	x_2	7	v	x_7
3	Lx	x_3	8	V	x_8
4	u	x_4	9	y	x_9
5	U	x_5	10	n	x_{10}

As symbols for the operations and subroutines, a system of "call-numbers" is used. These alphabetic characters represent the class of subroutines. Following Dr. Wilkes' example, these symbols are partially phonetic; that is, a = arithmetic, t = trigonometric, and x = exponential; amc = arithmetic, multiplication by a constant, x-e = e^{-u},ts0 = trigonometric, sine. Placed with the call-numbers, n, f, or s, indicates normal, floating, or stated (fixed) decimal point. Other letters and digits indicate radians or degrees for angles, complex numbers, etc. These call-numbers are listed in the catalogue together with the order in which arguments, controls, and results are to be stated. The general rules for the description of an operation are:

1. call-numbers,
2. number of operation,
3. arguments in order of appearance in formula, variables preceding constants,
4. controls, normal exit if altered, followed by alternate exits in order of appearance in subroutine,
5. results, in order of appearance.

All exceptions to the general rules are listed in the catalogue.

The problem has been reduced to computer information. The exact positions of characters in words as submitted to the UNIVAC has been omitted since it hardly seems of general interest. The preparation of information might be called creating a "multiple-address code", by which any number of arguments may enter an operation, to produce any number of results, and to proceed directly to the next operation unless routed to any one of several other operations.

Subroutines

Each subroutine in the library is expressed in coding relative to its entrance line considered as 001. They are, in general, programmed and coded for maximum accurary and minimum computing time. They may store within themselves constants peculiar to themselves. They may also make use of certain "permanent constants" read in with every program. These permanent constants occupy a reserved section of the memory and are called for by alphabetic memory locations, a trick, at present peculiar to UNIVAC. Thus, these addresses are not modified in the course of positioning the subroutine in a program. They include such quantities as $1/2\pi$, $\pi/4$, $\log_{10}e$, ± 0, .2, .5, and the like.

Each subroutine is preceded by certain information, matching and supplementing that supplied by the mathematician:

1. call-number;
2. arguments, the destination of the arguments within the subroutine, expressed in the relative coding of the subroutine;
3. non-modification indicators locating constants embedded in the subroutine which are not to be altered;
4. results, the positions of the results within the subroutine, expressed in relative coding.

Each subroutine is arranged in a standard pattern.

Entrance line - The first line of a subroutine is its entrance line, thus in relative coding it is number one. It is the first line of the subroutine transferred to a program, and it contains an instruction transferring control to the first action line.
Exit lines - The second line of a subroutine is its normal exit line. This contains an instruction transferring control to the line following the last line of the subroutine. Unless an alternate transfer of control is desired, all exits from the subroutine are referred to the normal exit line. Alternate exit lines, involving transfers of control from the usual sequence, follow the normal exit line in a predetermined order as listed in the catalogue.
Arguments - The exit lines are followed by spaces reserved for the arguments arranged in predetermined order.
Results - The results, also in specified order, follow the arguments.
Constants - The results are followed, when possible, by any arbitrary constants peculiar to the subroutine. When the subroutine has been compounded from other subroutines, groups of constants may also appear embedded in the subroutine. These are cared for by the non-modification information.

The first action line appears next in the subroutine. Its position in the relative coding is defined by the entrance line. No instruction line may precede this line.

The sequence assigned to the entrance and exit lines, arguments, results, and constants is arbitrary. It is convenient. All that is required is that a sequence be established and that the computer recognize this sequence.

For convenience in manipulation, a certain number of elementary subroutines have been combined to form a sub-library. These include

a =	arithmetic
b =	transfer of data
c =	counters
h =	hyperbolic functions
i =	input routines
l =	logarithmic functions
o =	output routines
p =	polynomials
r =	roots and fractional exponents
t =	trigonometric functions
u =	control transfers
w =	storage routines
x =	exponential functions
y =	editing routines

As subroutines are added to extend the library, it becomes more useful and programming time is further reduced.

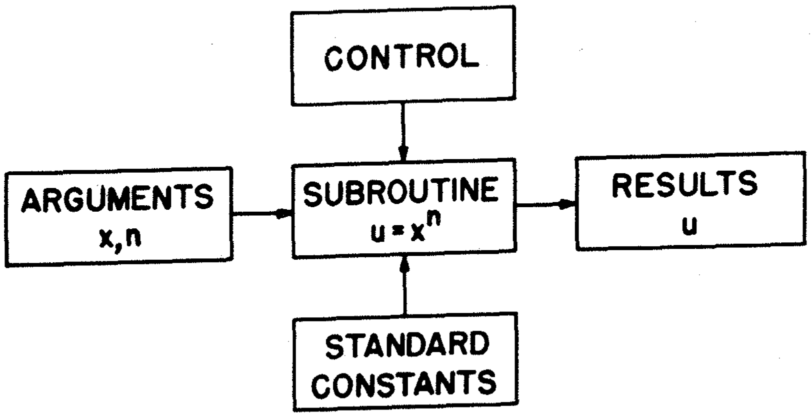

Fig. 8 - OPERATION

$y = e^{-x^2} \sin cx$

OPERATION NUMBER	OPERATION	ARGUMENTS	RESULTS	CONTROL
0	TRANSFER bOi	0, .01, .99, 2, .5 I (1, 2, 3, 4, 5)	$x, \Delta x, L_x$, n, c 1, 2, 3, 10, 6	
1	x^n apn	x, 2 1,10	$u = x^2$ 4	
2	e^{-u} x-e	u 4	$U = e^{-u}$ 5	
3	c⊗ amc	c, x 6, 1	u = cx 7	
4	sin v ts0	v 7	V = sin v 8	
5	⊗ am0	U, V 5, 8	y = UV 9	
6	EDIT yrs	x, y 1, 9	$\bar{x}, \bar{y}$ 0 (1, 2)	
7	$\oplus \rightarrow L$ aaL	$x, \Delta x, L_x$ 1, 2, 3	$x + \Delta x \rightarrow x$ 1	$x \leq L_x \rightarrow 1$, $x \leq L_x \rightarrow 8$ 8, 1
8	STOP ust			

Fig. 9 - EXAMPLE

Indeed, the day may come when the elementary subroutines are rarely used and the computer information will contain but seven or eight items calling into play powerful subroutines.

Construction of Subroutines

It is not necessary, nor is it advisable, that the inexperienced programmer tamper with the coding within a subroutine. It is usually minimum latency coding using every trick and device know to the experienced programmer. It has been tested by operation on the computer. However, in order to speed the original construction, on paper, of the elementary routines, kernel routines and threading routines have been devised.

A kernel routine computes a mathematical function or carries out an elementary process for a limited range of the variable concerned; for example, sin x, for $0 < x < \pi/4$ and 10^{-x} for $0 < x < 1$. A kernel routine is always entered and left by way of a threading routine.

Threading routines, incomplete without kernels, remove from the arguments and store, such quantities as algebraic signs, integral parts, and exponents, deliver the reduced arguments to the kernel routine, receive results from the kernel, and adjust algebraic signs and exponents. For example, the threading routines for sin y remove the algebraic sign of y, reduce y by multiples of 2π, reduce the remainder to a quantity x less than $\pi/4$, store the information and select the sin x or cos x kernel routine. The kernel routine returns sin x or cos x. The threading routine adjusts the sign, exponent, and decimal point completing the computation.

Threading routines recognize and give special treatment to such values as zero and infinity, and provide signals and printed information when the capacity of the computer is exceeded.

An elementary subroutine consists of a threading routine accompanied by one or more kernel routines. Hence, the threading routines are similar to the subroutines in form having at the beginning an entrance line, exit lines, (usually undetermined until the kernel routine is supplied), arguments, results, and constants. At the end of a threading routine are certain lines prepared to "overlap" the first section of the kernel routine. This overlap contains

1. the entrance line of the kernel routine;
2. the exit line of the kernel routine set-up by threading routine;
3. arguments; and
4. results.

Compiling Routines Type A are designed to select and arrange subroutines according to information supplied by the mathematician or by the computer. Basically, there is but one Type A routine. However, since the UNIVAC code contains instructions transferring two neighboring quantities simultaneously, a second compiling routine has been designed to care for floating decimal, complex number, and double precision programs. For each operation listed by the mathematician, a type A routine will perform the following services:

1. locate the subroutine indicated by the call-number;

2. from the computer and subroutine information combined with its record of the program, fabricate and enter in the program the instructions transferring the arguments from working storage to the subroutine;

3. adjust the entrance and normal exit lines to the position of the subroutine in the program and enter them in the program;

4. according to the control information supplied by the programmer, adjust alternate exit lines and enter them in the program (this process involves reference to the record);

5. according to the control information supplied with previous operations adjust auxiliary entrance lines and enter them in the program;

6. modify all addresses in the subroutine instructions and enter these instructions in the program;

7. according to information supplied by the subroutine, leave unaltered all constants embedded in the subroutine and transfer them to the program;

8. from the computer and the subroutine information fabricate and enter in the program the instructions transferring the results to

9. maintain and produce a record of the program including the call-number of each subroutine and the position of its entrance line in the program.

The compiling routines also contain certain instructions concerning input tapes, tape library, and program tapes, peculiar to the UNIVAC. All counting operations such allocation of temporary storage and program space, and control of input and output are carried on steadily by the compiling routine. Stated bluntly, the compiling routine is the programmer and performs all those services necessary to the production of a finished program.

Compiling Routines of Type B, will for each operation, by means of "task routines", replace or supplement the given computer information with new information. Thus, compiling routine B-1 will, for each operation, copy the information concerning that operation and call in the corresponding task routine. The task routine will generate the formula, and derive the information, necessary to compute the derivative of the operation. Compiling routine B-1 then records this information in a form suitable for submission to a Type A routine.

Since information may be re-submitted to a type B routine, it is obvious that in order to obtain a program to compute $f(x)$ and its first n derivatives, only the information defining $f(x)$ and the value of n need be given. The formulas for the derivatives of $f(x)$ will be derived by repeated applications of B-1 and programmed by a type A routine.

It is here that the question can best be answered concerning a liking for or an aversion to subroutines. Since the use of subroutines in this fashion increases the abilities of the computer, the question becomes meaningless and transforms into a question of how to produce better subroutines faster. However, balancing the advantages and disadvantages of using subroutines, among the advantages are

1. relegation of mechanical jobs such as memory allocation, address modification, and transcription to the UNIVAC;

2. removal of error sources such as programming errors and transcription errors;

3. conservation of programming time;

4. ability to operate on operations;

5. duplication of effort is avoided, since each program in turn may become a subroutine.

Only two disadvantages are immediately evident. Because of standardization, a small amount of time is lost in performing duplicate data transfers which could be eliminated in a tailor-made routine. In base load problems, this could become serious. Even in this case, however, it is worthwhile to have UNIVAC produce the original program and then eliminate such duplication before rerunning the problem. The second disadvantage should not long remain serious. It is the fact that, if a desired subroutine does not exist, it must be programmed and added to the library. This will be most likely to occur in the case of input and output editing routines until a large variety is accumulated. This situation also emphasizes the need for the greatest generality in the construction of subroutines.

Several directions of future developments in this field can be pointed out. It is to be hoped that reports will be presented on some of them next September.

More type A compiling routines will be devised; those handling commercial rather than mathematical programs; some special purpose compiling routines such as a routine which will compute approximate magnitudes as it proceeds and select sub-routines accordingly. Compiling routines must be informed of the average time required for each sub-routine so that they can supply estimates of running time with each program. Compiling routines can be devised which will correct the computational procedure submitted to produce the most efficient program. For example, if both $\sin \theta$ and $\cos \theta$ are called for in a routine, they will be computed more rapidly simultaneously. This will involve sweeping the computer information once to examine its structure.

Type B routines at present include linear operators. More type B routines must be designed. It can scarcely be denied that type C and D routines will be found to exist adding higher levels of operation. Work is already in progress to produce the formulas developed by type B routines in algebraic form in addition to producing their computational programs.

Thus by considering the professional programmer (not the mathematician), as an integral part of the computer, it is evident that the memory of the programmer and all information and data to which he can refer is available to the computer subject only to translation into suitable language. And it is further evident that the computer is fully capable of remembering and acting upon any instructions once presented to it by the programmer.

With some specialized knowledge of more advanced topics, UNIVAC at present has a well grounded mathematical education fully equivalent to that of a college sophomore, and it does not forget and does not make mistakes. It is hoped that its undergraduate course will be completed shortly and it will be accepted as a candidate for a graduate degree.