Project Whirlwind
Servomechanisms Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: TECHNIQUES FOR USING STANDARD AUTOMATIC SUBROUTINES

To: Mathematics Group

From: C. W. Adams

Date: February 10, 1950

Abstract: Standard automatic subroutines are programs for the evaluation of frequently needed functions or for the performance of routine computational chores. Such subroutines, which are intended primarily for use as a part of a larger main program, will be kept on permanent file in the slow-speed memory of a computer and will be inserted into high-speed memory along with any program of which they are to be a part. A subroutine, if it is to be referred to from several different parts of a main program, must be prepared for each use by the insertion of new addresses into some if its orders.

In the present note, various preparation techniques are discussed. Most promising is a deferred preparation, in which the only preparatory order needed in the main program is a transfer of control order, all of the necessary changes in the subroutine being performed by orders in the subroutine.

In the Whirlwind computer the address of the register next after the register containing a transfer of control is stored automatically in the A register by the transfer of control (sp) order. This register, next after the one containing the sp order, is a prearranged location characteristic of one and only one sp order, and as such this register and the ones following it can be used to store any addresses which must be available to the subroutine in making the necessary preparations. That is, the correct return address and the addresses of registers which are to contain all necessary extra addresses can all be deduced within the subroutine with no assistance from the main program.

Introduction

The simplest computer imaginable would need to have only a few basic abilities -- in fact, the ability to (1) sense and (2) complement a selected binary digit would be sufficient. Unfortunately, many repeated

applications of these basic operations would be required to perform a single complete addition since the addition of each pair of digits and each carry would have to be ordered separately; so the program for any sizable problem would be unwieldy at best. In designing a computer, therefore, one builds a control and an arithmetic element of at least enough complexity to be able to perform many of the elementary arithmetic operations in response to single orders in the program. The number of elementary operations so built in may influence the speed, effective memory capacity and convenience in use of the computer, but it in no way limits the capability of the machine. For just as addition can be programmed using only a single-digit sense-and-complement order, any of the more complicated numerical operations can be programmed using the simpler built-in operations.

The Whirlwind computer has built into it the abilities to add, subtract, multiply, divide, find magnitude, and point off (find the characteristic of the $\log_2 x$ for any given x), using numbers within the prescribed range. Other operations, such as the evaluation of $\sqrt{x}$, $\log x$, e^x, $\sin x$, etc., are not built in. The line between included and excluded operations is drawn, as it is in all such cases, by a compromise based on such considerations as the flexibility, convenience, speed, economy of construction, simplicity, and ease of maintenance desired; these considerations hinge in turn on the intended purpose of the machine. The decision is made less difficult by the comparative ease with which the non-automatized operations can be synthesized from the elementary ones and made to seem almost automatic.

Purpose

The intention of this report is to combine and bring up to date under one cover the pertinent information on the means of accomplishing programmed automatization of the non-elementary but nonetheless common numerical operations. In the present report, a working knowledge of coding for the Whirlwind computer is presupposed. The Appendix contains a summary guide to coding and a brief but exact description of the Whirlwind order code as of January 1950.

Standard Automatic Subroutines

Programs that are commonly used as part of a larger program are called subroutines. Because the subroutines described in this report will be so arranged that they seem to be built in, these subroutines are called automatic. Only one or at most a few of the many possible variants of the subroutines for each function will be kept permanently available (in the case of sin x, for example, different routines would be provided at least for (1) x in revolutions, (2) x in radians, (3) x in radians times some scale factor.) Such subroutines will then be called standard

automatic subroutines for the evaluation of the given functions. In this note, techniques for preparing to use the subroutines are discussed at length.

Standard automatic subroutines will be kept on permanent file on the slow-speed film storage of the Whirlwind computer and each will be inserted into high-speed storage along with any program in which the particular subroutine is needed. Some indication of the necessary subroutines and of the addresses of the storage registers to be occupied by each subroutine will be given as part of each main program. That is, in writing each program, the programmer will be able to select for use in his program any of the standard automatic subroutines and he will be able to designate the storage registers in which each of the selected subroutines is to be stored, subject only to the condition that his assignments are compatible with the length of each subroutine so that there are enough consecutive storage registers available for each of the subroutines.

According to the present plans for Whirlwind I, all of the standard automatic subroutines will have been written, converted to binary form, and stored in some order on one (or more if necessary) roll of film called a library film. Each subroutine will have been written under the assumption that it is to be stored beginning at register #1024. All the necessary constants, except possibly some universal constants like 1/2, will be included as a part of the subroutine, stored in registers immediately following the last order of the subroutine. The purpose of writing each subroutine starting at register 1024 is to permit easy discrimination by the computer between those orders whose address sections refer to other parts of the subroutine and those orders whose addresses are irrelevant or refer to something else (such as a number of shifts, an address of an external device, or an address of some universal constant, all of which would normally be less than 1024).

The actual insertion of a program into the computer will probably be done with the aid of three preliminary routines, all of which will presumably be stored at the beginning of the library film and will be put into the high-speed storage of the computer by means of the ri operation. The main program to be performed will have been typed out in some normal fashion on an automatic typewriter which at the same time prepares a perforated tape. (Flexowriter equipment will be used for this purpose.) Each character (i.e., each of the 50 keys and controls on the typewriter) has a six-digit binary representation on the tape and this binary-coded information can be translated in the computer to the proper binary form and stored in the proper registers by means of a suitable conversion program. The typewritten form of each new program will also have, probably at the end, some indication of the subroutines needed and the addresses assigned to each. The conversion routine, the first of the three preliminary

routines, will then turn this information, properly translated, over to the second preliminary routine, a library selection routine, which will select the desired subroutines from the library film. The third subroutine (the adaptation routine) will take each subroutine and change the addresses as necessary to adapt the subroutine to its assigned place in storage. These preliminary routines will be discussed in a subsequent note.

A library of subroutines is actually being built up, starting with the common function evaluations, most of which have already been prepared in preliminary form and published (cf. E-170, C-70, C-77 for e^x and log x, sin x and cos x, and $\sqrt{x}$ respectively). A note which will contain revised and "final" forms of these and other subroutines is also forthcoming.

Classification of Subroutines

Standard automatic subroutines can be classified according to the amount of information which must be exchanged between the subroutine and the main program each time the subroutine is used. Obviously, every subroutine must be supplied with the proper return address -- the storage address of the next order in the main program to which control is to be returned at the completion of the subroutine. Aside from the return address, many subroutines such as those for the evaluation of x, log x, e^x, sin x, etc., need only to be given the value of x and need only to supply the value of the desired function. Since the quantity x and the resulting function of x will in most cases occupy only a single register each, the simplest procedure, apparently, is for the main program to put the quantity x into the Accumulator (AC) just before transferring control to the subroutine and for the subroutine to put the resulting function of x into AC just before returning control to the main program. Thus in this case no storage address, other than the return address, needs to be exchanged. Subroutines of this type, requiring the exchange of no addresses, will be said to be zero-address subroutines.

Some subroutines require the exchange of some number other than the quantity x and the result. For example, the quantity x or its result may be double-length -- i.e., require two registers to accommodate it because of its magnitude or its precision or both. (Frequently, however, a two-register result such as a number and a scale factor will be obtained from a zero-address subroutine since the result can easily be stored with the number in AC and the scale factor in some predetermined register, chosen once and for all.) Or, as another example, a subroutine intended to shift the contents of AC and BR left without roundoff must be supplied with the number of shifts to be performed. In such cases it is necessary for the main program to supply an address, over and above the return address, to

the subroutine -- that address being either the address at which some necessary quantity will be found or at which some result is to be stored. Subroutines of this type, requiring that one address be exchanged, will be called one-address subroutines.

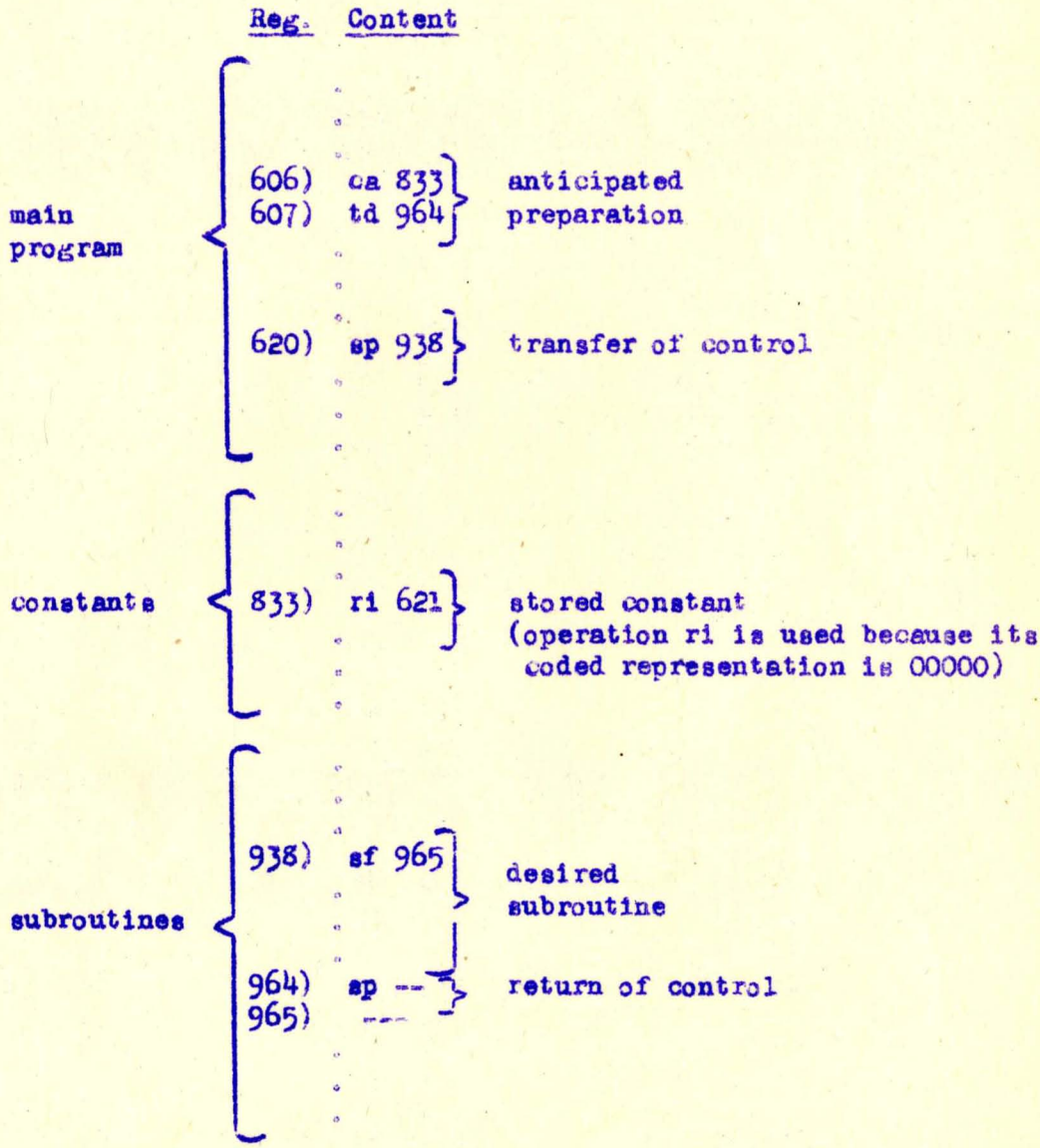
Similarly, other subroutines require the exchange of two, three or more orders. Thus, finding a quotient plus a remainder requires two addresses, one at which the divisor is stored and one at which the remainder is to be stored, with the dividend being put into AC and the quotient appearing in AC. The double-length arithmetic operations (addition, subtraction, multiplication, division) generally require three addresses, one for each of two operands and one for the result, since the three quantities involved are each double-length and cannot be stored in AC. In the case of double-length numbers, three pairs of addresses, six in all, are actually required, but it is assumed that the two halves of a double length number are stored in consecutive registers so that the second address of a pair can always be deduced from the first address and is not therefore a separate piece of information. As before, the number of addresses, exclusive of the return address, characterize the subroutine so that the quotient-and-remainder program is two-address while the double length operations are three-address subroutines.

Automatization of Zero-Address Subroutines

Fundamentally a subroutine is a set of orders which is used in several different parts of a main program but which is only to be put in one place in storage. The problem of automatizing the subroutine is just the problem of how to permit the subroutine to be effectively inserted into the main program by unconditional transfers of control from the main program to the subroutine and back again. By definition, the zero-address subroutines require the exchange of no address except the return address.

Suppose, for example, that the programmer has requested that a subroutine for the calculation of the square root of a number be stored starting in register 938. Suppose further that, at the completion of the main program order stored in register 619, the quantity x is in AC and that the square root of x is wanted. Then the order stored in register 620 might be sp 938 which would transfer control to the start of the square root subroutine. At the end of the square root subroutine is another sp order, which in this case should be sp 621, to return control to the proper point in the main program. This address 621, the return address, must be supplied from somewhere. It obviously cannot be simply written in once and for all, for the subroutine will probably be referred to from several different places in the main program and the return address will differ in each case.

One way to supply the return address would be to store the return address, which is known in each case, in some predetermined register. Then in the main program, before the sp 938 order, one could clear and add that address and transfer it, using the td operation, to the register containing the sp order at the end of the subroutine. This procedure requires at least two extra orders and one extra register of constant storage for each place in which the subroutine is to be inserted into the program. In addition it presupposes anticipation by the programmer who must be sure that the ca and td sequence is inserted in the main program before the quantity x is formed in AC, it being assumed that x is to be in AC when control is transferred to the subroutine.



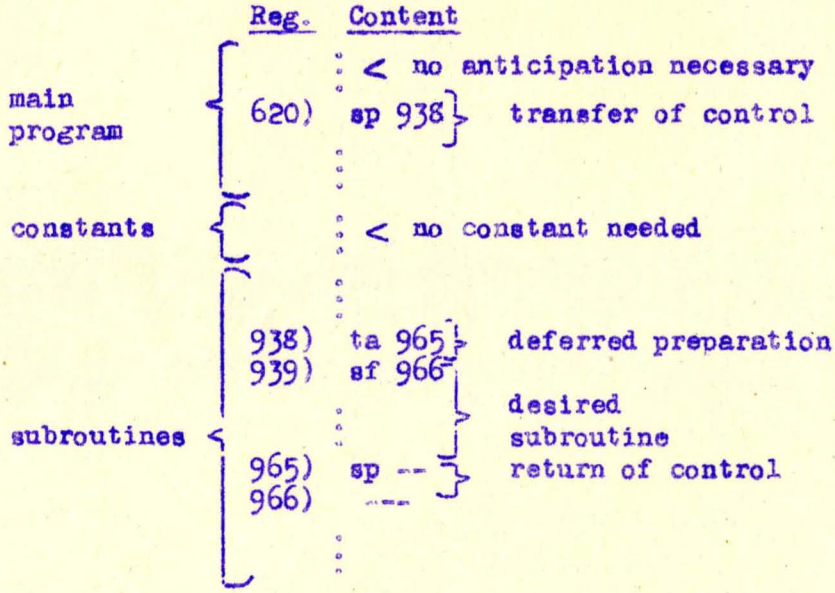
ANTICIPATED PREPARATION FOR
A ZERO-ADDRESS SUBROUTINE

This anticipated preparation of the return address is possible in almost any computer. But in Whirlwind I, the addition of one simple feature to the sp operation and the addition of one new operation, ta, has removed the need for the clumsy procedure just described. The function of these two orders is as follows:

sp x - Transfer control unconditionally to register x. Assuming that the sp x order is stored in register y, store the quantity y+1 (which is obtained from the program counter before control is transferred) in the last 11 digit positions of AR.

ta x - Transfer the last 11 digits from AR to the last 11 digit positions of register x, leaving the first 5 digits unchanged.

Thus in the example of the previous paragraph, the return address 621 would be stored in AR, without affecting the contents of AC, by the sp 938 order which was stored in register 620. Then the first order in the subroutine would be a ta order which would transfer the return address to the last order, the sp order, at the end of the subroutine. The intervening orders of the subroutine would determine the square root of the number in AC and leave the result in AC. In this way the red tape involved in using the zero-address subroutines is reduced to practically nothing. The square root of the number is found by simply inserting the order sp 938 into the program whenever the square root of the contents of AC is wanted. Using the zero-address subroutines requires no more thought or effort than using a single operation built into the computer.



DEFERRED PREPARATION FOR
A ZERO-ADDRESS SUBROUTINE

Automatization of One-Address Subroutines

A one-address subroutine can be automatized in much the same way as a zero-address subroutine. The only difference is, of course, that some storage address in addition to the return address must be made available to the subroutine from the main program. Consider, for example, the subroutine for shifting left without rounding off. It is assumed that the number to be shifted is in AC and BR when control is transferred to the subroutine and that the shifted result is to be left in AC and BR at the end of the subroutine.

The subroutine itself could be

Reg.	Content	
718)	ts 751	} store content of AC
719)	sl 15	} store content of BR
720)	ts 752	
721)	ca 751	} shift and store the original content of AC
722)	sl n	
723)	ts 751	
724)	ca 752	} shift content of BR
725)	mh 753+n	
726)	ad 751	} add in shifted content of AC
727)	sp --	} return of control
751)	---	
752)	---	
753)	2^{-15}	
	⋮	
753+n)	2^{n-15}	
	⋮	
767)	2^{-1}	

EXAMPLE: A SUB-ROUTINE TO SHIFT LEFT WITHOUT ROUNDOFF

Actually, such a subroutine would be valid only for n in the range $0 \leq n \leq 14$ and consequently the standard automatic subroutine might well be slightly more complicated and more general. The subroutine just given is, however, satisfactory as an example of various preparation techniques.

The most obvious technique is anticipated preparation of the extra address, using deferred preparation of the return address. In the example given, the number of shifts, n, must be inserted in the sl n order in register 722 and the address 753+n must be inserted in the mh 753+n order in register 725. Note that although two different addresses are needed, one may readily be deduced from the other and consequently the subroutine is classed as a one-address subroutine. The return address must be inserted in register 727, but this can be handled as in the zero-address technique. The result is

	Reg.	Content	
main program	223)	ca 690	anticipated preparation of the extra address (es)
	224)	td 722	
	225)	ad 728	
	226)	td 725	
	240)	sp 717	transfer of control
constants	690)	ri n	stored constant address
subroutines	717)	ta 727	deferred preparation of return address
	718)	ts 751	desired shift-left subroutine
	719)	sl 15	
	720)	ts 752	
	721)	ca 751	
	722)	sl --	
	723)	ts 751	
	724)	ca 752	
	725)	mh --	
	726)	ad 751	
	727)	sp --	return of control
	728)	ri 753	constant address needed to prepare address for the mh order

ANTICIPATED PREPARATION OF
A ONE-ADDRESS SUBROUTINE

A second technique is the use of a deferred preparation of the extra address as well as of the return address. This can be accomplished by storing the extra address in some prearranged location so that the preparation orders can be put into the subroutine, thereby obviating the duplication of the preparation orders in the main program. It is not satisfactory to simply decide that the address should be stored in some one register, such as in 690 as it was in the example; for since the address will presumably be different each time the subroutine is used, the main program would still be required to transfer the address to some given location such as 690. (Although in the case at hand, in which the quantity n is needed in two different orders, some economy of orders could indeed be made by letting the main program put only the first address into the subroutine and letting orders in the subroutine form the second address from the first.)

In the deferred preparation technique to be described, the extra address is stored in a location so chosen that each reference to the subroutine from the main program designates uniquely its own prearranged location. Thus far in this discussion, the return address has been the address of the register next after the register which contains the transfer of control (the sp order). However, this register, next after the one containing the sp order, is a prearranged location characteristic of one and only one transfer of control point, and as such this register could be used to store the necessary extra address. The correct return address would then be the address of the register second after the transfer of control point.

	Reg.	Content	
main program			< no anticipation necessary
	240)	sp 717	transfer of control
	241)	ri n	the necessary address, stored in a dummy order
constants			< no constants needed, except those directly associated with the subroutine
subroutines	717)	ta 723	preparation for deferred preparation of extra addresses
	718)	ta 733	partial deferred preparation of the return address
	719)	ts 751	storing the contents of AC and BR (start of the shift-left subroutine)
	720)	sl 15	
	721)	ts 752	
	722)	ao 733	completion of deferred preparation of the return address
	723)	ca --	deferred preparation of the extra address (es)
	724)	td 728	
	725)	ad 734	
	726)	td 731	

(continued on next page)

	Reg.	Content	
subroutines (continued)	727)	ca 751	completion of the desired shift-left subroutine
	728)	sl --	
	729)	ts 751	
	730)	ca 752	
	731)	mh --	
	732)	ad 751	
	733)	sp --	return of control
	734)	ri 753	constant address needed to prepare address for the mh order
		⋮	

DEFERRED PREPARATION OF A ONE-ADDRESS SUBROUTINE

Automatization of Several-Address Subroutines

The generalization to several addresses of the preparatory techniques just described for one-address subroutines is almost trivial and little need be said. The anticipated preparation works in exactly the same fashion except that more than one address must be stored in constant storage and must then be transferred by orders in the main program to the subroutine. The deferred preparation is likewise almost unchanged; the several addresses are simply stored as dummy orders in consecutive registers immediately following the sp order in the main program, and the return address is simply the address of the register next after all of the several dummy orders.

History of the Discontinued "Automatic Subprogram" Operations

One of the most important set of three-address subroutines for a computer with comparatively short register length is the double-length operation subroutines. Some time ago (cf. M-111, pps. 8-10, dated October 6, 1947) it was proposed that a set of five special operations be incorporated into the design of the Whirlwind computer primarily to facilitate work with double-length numbers. The first of these operations, designated by as, would (1) transfer the contents of AC bodily into BB and (2) transfer the as order itself into AC. Three of the operations were logically identical; these operations, designated by ax, ay, az, would (1) transfer the ax (or ay or az) order itself into AR, (2) transfer the return address from the program counter to register 2047, and (3) transfer control unconditionally to some preselected -- i.e., wired in -- storage address, three different addresses being selected, one by ax, one by ay, one by az. A fifth operation, logically almost equivalent to the present ta operation but designated by ro, permitted the contents of AR to be read into AC.

The intended function of these operations is best illustrated by an example. Supoose a double-length addition subroutine is to be used and that the augend is stored in registers 618 and 619, the addend in registers 712 and 713, and the sum is to be stored in registers 832 and 833. Then the preparatory orders would be

as 618
as 712
ax 832

The first order would put the address 618 into AC; the second order would shift the address 618 into BR and put the address 712 into AC; the third order would put the address 832 into AR, would transfer control to a preselected position (x), and would store the return address in register 2047. Then the subroutine for double-length addition, stored beginning in register x, would proceed to unstack the various addresses stored in AC, AR and BR and supply them where needed in the subroutine, would deduce from the given addresses the second address of each pair (e.g., 619 = 618 + 1), would transfer the return address from register 2047 to the return of control (sp order) at the end of the subroutine, and would then perform the double-length addition.

The so-called "automatic subprogram" operations were eliminated from the Whirlwind I order code (cf. E-235, May 6, 1949). The reason for mentioning them here is threefold. First, these operations were referred to in a number of notes on programming techniques written in 1948 and it seems advisable to take cognizance of them for the benefit of anyone who has already or may yet encounter references to them in the literature of Project Whirlwind. Second, these operations point out at least one way in which special operations can be built into the machine to facilitate particular applications. Third, the method used is fundamentally a good one and provides a standard for comparison with other techniques. The reason that the automatic subprogram operations were dropped was simply that their value did not justify their existence when compared to the possible value of other special built-in operations. It should be noticed that there would be almost no gain, in fact less than none in some cases, in using the automatic subprogram operations in preparing for zero- or one-address subroutines because (1) only three locations (x, y and z) and consequently only three different subroutines can be used and (2) the use of the arithmetic element in storing addresses is obviated by the fact that in many cases the numbers themselves can be stored in AC even more effectively than their addresses.

Evaluation of the Preparation Techniques

There are several criteria by which a programming technique should be evaluated. The most important of these criteria can be summed up in the form of four questions.

(1) Is the technique easy to learn and to use?
(2) Does the technique reduce coding and manual preparation time?
(3) Does the technique reduce the storage capacity needed to solve the problem?
(4) Does the technique reduce the computing time needed to solve the problem?

In evaluating the three methods discussed in this report, one might prepare a table of comments on their relative merits. Of course, the use of special orders is not a technique of any practical importance in the Whirlwind or any other computer at present, since the orders do not exist; but it is well to keep the possibility in mind. It should also be noted that deferred preparation of a return address is only possible in a computer such as Whirlwind having the appropriate orders (sp and ta) in its code. But deferred preparation of extra addresses is possible in any digital computer, even if the order code requires use of anticipated preparation of the return address, for once the return address is available to the subroutine, the addresses of registers containing the extra addresses are also available.

EVALUATION OF PREPARATION TECHNIQUES

criterion	anticipated preparation	deferred preparation	use of special "automatic subprogram" orders
ease in learning and applying	quite easy to learn since no special technique of coding is needed; cumbersome in use	requires special knowledge, but once learned is easy to use	requires some special knowledge, but once learned is probably the easiest to use. Lacks generality since it cannot be applied to many subroutines.
reduction in coding and manual preparation time reduction in required storage capacity	wasteful and cumbersome	quite efficient since only the essential addresses need be inserted (including the address of the desired subroutine)	most efficient for double-length numbers, since the address of commonly used subroutines does not need to be specified by the main program, but has no advantage over deferred preparation in many applications
reduction in the required computing time (actually, use of subroutines necessarily increases computing time compared to not using subroutines at all)	uses two orders to prepare the return address, which is poor; uses two orders to prepare each distinct extra address, which is good.	uses one order to prepare the return address, which is good; uses four orders to prepare each distinct extra address, which is poor.	uses two orders to prepare the return address, which is poor; uses two, plus (to get an address out of BR), orders to prepare each distinct extra address, which is fair.

Conclusions

The deferred preparation technique will be used in connection with all standard automatic subroutines for Whirlwind I. By this method, only one order is needed to bring about the evaluation of a common function, and only one extra register is needed for each extra address which is to be supplied.

Use of the various preparation techniques is shown in the following example, where the program is not an example of efficient coding but merely illustrates the thoughtless, brute force way in which results can be obtained.

Required to evaluate $(e^x \sin y - e^y \sin x)2^8$

where it is known that $0 > x > -1$
$0 > y > -1$
$2^{-8} > e^x \sin y - e^y \sin x > -2^8$

Suppose the following subroutines are available:

Evaluation of e^x for $-1 < x \leq 0$, first order stored in register A

Evaluation of sin x for $-1 < x < 1$, first order stored in register B

Double-length subtraction, first order stored in register C

A subroutine to take a double-length number, shift it left n times and put it back in the same pair of registers, first order stored in register D

Other registers are assigned as follows:

register X contains x

register Y contains y

registers T1, T2, etc. are consecutive registers available for temporary storage

The program then is: (asterisks indicate use of a standard subroutine)

```
    ca X
*   sp A    } find and store e^x
    ts T1
    ca Y
*   sp B    } find sin y
```

```
    mh T1
    ts T1
    sl 15      } form and store 30-digit product e^x sin y
    ts T2
    ca Y
*   sp A       } find and store e^y
    ts T3
    ca X
*   sp B       } find sin x
    mh T3
    ts T3
    sl 15      } form and store 30-digit product e^y sin x
    ts T4
*   sp C       subtract the second product from the first
    ri T1      address of minuend
    ri T3      address of subtrahend
    ri T1      address of difference
*   sp D       shift the result left 8 times
    ri 8       number of shifts required
    ri T1      address of operand
```

Thus it is seen that such operations as the evaluation of e^x or sin x and subtracting or shifting double-length numbers can be programmed almost as easily as if they were built in to the computer, using deferred preparation of standard automatic subroutines.

Signed C. W. Adams
C. W. Adams

Approved R. R. Everett
R. R. Everett

CWA/lfu/aec

Attached: A Short Guide to Coding.