SILLIAC PROGRAMMING MANUAL

=====THE ADOLPH BASSER COMPUTING LABORATORY

SCHOOL OF PHYSICS

THE UNIVERSITY OF SYDNEY

ADOLPH BASSER

DIGITAL COMPUTING LABORATORY

SILLIAC PROGRAMMING

A Guide to the Preparation of Problems
for Solution by
the Automatic Digital Computer
in the Adolph Basser Computing Laboratory
within the School of Physics
of the University of Sydney

Second Edition

SYDNEY
January, 1959.

PREFACE

The Adolph Basser Computing Laboratory has been set up with funds donated to the Nuclear Research Foundation by Dr. Adolph Basser, C.B.E., Governor of the Foundation. The Laboratory is equipped with a high speed automatic computer which is an almost exact copy of the automatic computer at the University of Illinois, the ILLIAC, and as it is the Sydney version of the ILLIAC, it has been called the SILLIAC.

One of the fortunate results of the generous assistance given by the Digital Computer Laboratory at the University of Illinois is that we in Sydney have been given a flying start by being able to take over the program library developed at Illinois. Moreover, we have been able to profit from the wide teaching experience of the group at Illinois as embodied in the ILLIAC programming manual; and, in fact, with minor alterations only, this handbook is a direct copy of the ILLIAC programming manual. We would like to express our gratitude here for permission to make this copy.

The original manual was the result of a group effort by many different people, and chapters were written by S. Gill, R. E. Meagher, D. E. Muller, J. P. Nash, J. E. Robertson, T. Shapin and D. J. Wheeler.

PREFACE TO SECOND EDITION

Towards the end of 1958, several new orders were introduced into the SILLIAC code. At the same time, the state of the old stencils was such that they could not be used for producing further copies of the first edition. Accordingly, the opportunity was taken to revise the first edition, many sections being expanded considerably. Contributors to this new edition include J. M. Bennett, B. A. Chartres, Miss Joy Elliott, B. E. Swire and R. Whitfeld. The typing of the stencils was carried out by Miss Lena Trachtenberg.

CONTENTS

CHAPTER 1
INTRODUCTION

In 1953 the Physics Department of the University of Sydney began to look into the possibility of acquiring an automatic digital computer. At the time, it was clear that the most satisfactory way of going about this was to copy a machine of proven design, so that as much use as possible could be made of existing experience.

The funds for setting up the organisation necessary for constructing a machine were made available by Dr. Adolph Basser, C.B.E., Governor of the Nuclear Research Foundation, and this organisation is now known as the Adolph Basser Computing Laboratory within the School of Physics of the University of Sydney.

At the time a careful survey was carried out of machines in operation throughout the world which might be suitable for carrying out the type of computation anticipated. The machine most suited to this class of work was the type operating at the Digital Computing Laboratory of the University of Illinois, the ILLIAC, and thanks to the co-operation of this Laboratory, circuit diagrams of this machine were made available. These, with only minor modifications, were used for constructing the machine which has become known as the SILLIAC (Sydney version of the University of ILLInois Automatic Computer).

The University of Illinois machine was one of two machines built at the University of Illinois Digital Computer Laboratory, the other having been constructed under contract for the U.S. Army. Work on these two machines was begun shortly after the foundation of the Laboratory in February, 1949, and the first machine to be completed, called the ORDVAC, has been in use at Aberdeen since February, 1952. The ILLIAC was completed in September, 1952.

The SILLIAC is an automatic electronic digital computer.

It is digital because it handles numbers as sets of digits which have discrete values, rather than as scale readings or measurements, which are continuously variable. Apparatus for handling digits is more complicated than that for handling continuous quantities, but it is capable of giving unlimited accuracy by using suitable numbers of digits.

The SILLIAC is electronic. In the last ten years electronic circuits for storing, transmitting, adding, subtracting, multiplying and dividing numbers in digital form at extremely high speeds have been devised. The addition of two numbers in the SILLIAC takes only about 75 microseconds.

Such speed is useless unless the machine can be made to go ahead on its own with many thousands of operations, without human intervention. The SILLIAC is therefore automatic, in the sense that it can be given orders telling it how to proceed, and will then act on these orders automatically.

In common with many other computers of a similar type, the SILLIAC contains the following five essential features:

(1) An arithmetic unit.

(2) A memory or store.

(3) Devices for the input and output of information (e.g. numbers) to and from the machine.

(4) Means for the transfer of information between the various parts of the machine.

(5) Means for the automatic control of the whole machine.

The arithmetic unit carries out the individual arithmetical

operations that make up every computation; it can be thought of as the electronic equivalent of a desk calculating machine. It is described in detail in Chapter 2.

The memory is needed because, in any lengthy calculation, numbers produced at early stages of the calculation are frequently required to be used at later stages; they must therefore be recorded or "remembered". The memory is capable of recording 1024 numbers. These can be recorded (i.e. transferred to the memory from the arithmetic unit) individually, as directed by the computer's control device, and recalled again individually in a similar way. The memory may be thought of as 1024 little boxes or locations, each accommodating one number, and labelled with the numbers 0 through 1023. The label of a location is called its address. A number in the memory is identified by the address of the location containing it.

Information enters and leaves the SILLIAC coded in the form of a pattern of holes in punched paper tape; there is a tape reader for input, and an automatic punch for output. There is also a teletypewriter which can be used to provide output from the machine directly in printed form. Several machines are available for preparing punched tape, copying it, comparing it, and producing printed versions of the information on it.

The problem of controlling the whole computer has been solved by stipulating that every individual operation that occurs within the machine must be one of a certain set of specified permissible operations, and that no two such operations can occur simultaneously. Thus the design problem was reduced to that of engineering the various permissible operations and arranging for them to be executed in any desired sequence. It is up to the user of the SILLIAC to specify the sequence of operations or

program, which the SILLIAC must execute to carry out his calculation.

Each permissible operation can be specified in a concise coded form called an order. The correspondence between the set of permissible operations and the set of orders which specify them is called the order code of the SILLIAC. It is given in detail in Chapter 3. A coded problem is called a program or routine.

The machine's control unit has the task of accepting orders one by one, and of causing the machine to carry out the operations specified according to the order code. If each order were taken by the control unit directly from a punched tape, then to make full use of the speed of the rest of the machine the tape would have to pass through the tape reader at about 200 miles per hour. Instead, the orders are recorded in the memory along with the numbers, so that the control unit merely has to take its orders from the memory, which it can do electronically at high speed. This is made possible by coding each order to look like a number. To be more precise, orders are stored in pairs, one pair to a memory location. The information contained in one memory location is often called a word, meaning either a number or an order pair. Of course, the more orders there are in the memory, the less room there is for numbers. Both orders and numbers are fed into the machine initially on punched tape.

Normally orders are obeyed by the control unit in the sequence in which they are stored in the memory, e.g.:

Left-hand order in location 6,
Right-hand order in location 6,
Left-hand order in location 7,
Right-hand order in location 7,
Left-hand order in location 8, etc.

Sometimes, however, this sequence is broken and the control unit starts over at some new position in the memory; this is called a transfer of control. There are special orders which cause this. There is also special provision for making a transfer of control depend on the value of some number obtained by the machine during the calculation. Thus the machine can be made to "choose" one of two or more alternative courses of action according to the way things happen to work out.

If control is transferred a few locations back in the memory, the machine will repeat the operations specified by the intervening orders. It is possible to cause this repetition to occur any number of times, leading to a cyclic behaviour of the machine. Practically every calculation which the machine performs contains several such cycles, often one inside another. In this way it often happens that the same order gets carried out many thousands of times, so that a few orders suffice to keep the machine busy for several minutes. If each order in the memory were to be carried out once only, the SILLIAC would get through them all in a quarter of a second (even if the memory contained nothing but orders). In practice, calculations vary in duration from a minute to a few hours.

The occurrence of cycles is one of the things that complicates the programming of a calculation. Another is the fact that, since orders are stored in the memory in the same form as numbers, they can be operated on and altered during the course of a calculation (at the behest of other orders) just as if they were numbers. All this makes possible some most interesting calculations; it can also make programming difficult.

Fortunately a coder can often, as described in Chapter 4, make use of bits of programming done by other people.

Thus, a typical program consists of a number of groups of orders, some written by the coder, others already available. The latter will be available in punched tape form, and can be copied mechanically onto the program tape along with the new orders. Tape preparation is described in Chapter 9.

When the whole tape for a particular program has been prepared, it can be placed in the tape reader of the SILLIAC. The SILLIAC reads the tape, forms the orders and numbers punched on it and stores them in the memory. When the program is in the memory, the machine begins to execute the orders, continuing until it comes to some particular order which causes it to stop. If the programming is correct, this is the end of the calculation. If there is a mistake in the programming, various things may happen; remedies are discussed in Chapter 8.

Somewhere in the program will be some orders which cause the machine to punch some output tape. This carries the results of the calculation. The program may also contain orders causing the machine to read more input tape, carrying data for the calculation.

The reading of most of the program tape is accomplished by the SILLIAC executing a particular set of orders called the Decimal Order Input (see Chapter 5) which is always punched at the beginning of every program tape and hence read into the machine before the rest of the tape. The Decimal Order Input not only assembles the program inside the machine; it also makes certain modifications and conversions, so that the way in which orders are represented when punched is somewhat different from their final form in the memory. The object is to make programming easier. It is important to remember that the written form of an order and the form which it assumes in the memory are not the same

thing. The relationship between the two is determined by the Decimal Order Input.

Remaining chapters of this manual are devoted to: the arrangement of calculations so that all the numbers encountered are the right size (Chapter 6); ways of programming certain types of simple tasks (Chapter 7); how to estimate the duration of a calculation (Chapter 10); a simplified coding scheme (Chapter 11); and how the program library is organized (Chapter 13). Finally, concise descriptions of the principal contents of this collection are given.

Starred sections in this handbook may be omitted at first reading. Moreover, anyone who wishes merely to carry out a small calculation with the aid of SILLIAC is advised to proceed directly to Chapter 11, where details of a simplified coding scheme will be found. Even if the reader intends to make extensive use of the machine, he should, at an early stage, code a practice example for SILLIAC with the help of this scheme--perhaps even before tackling the rest of the handbook.

corresponds to a division by two. For example,

0.0111	shifted left is 0.1110; 7/16 x 2 = 7/8
0.1110	shifted right is 0.0111; 7/8 ÷ 2 = 7/16
1.0010	shifted right is 1.1001; -7/8 ÷ 2 = -7/16
1.0100	shifted left is 0.1000; -3/4 x 2 = 1/2 because of overflow.

It should be noted that the sign digit is propagated when the right arithmetical shift is executed. The SILLIAC arithmetic unit has two shifting registers, each capable of executing both left and right shifts. In addition to arithmetical shift, other variants of the shift operation are available. These are described in the next chapter (q.v.).

Clearing to zero involves setting all digits of a number to zero; the corresponding arithmetic value is zero.

2.3 THE SILLIAC ARITHMETIC UNIT. The structure of the SILLIAC arithmetic unit is shown in Figure 2.2. The arithmetic unit is composed of two shifting registers, the accumulator A and the quotient register Q, and one non-shifting register, the number register R^3. Also required are the complement gate and the adder. The A and Q registers are the only registers to which the programmer has direct access; the number register R^3 is used to hold temporarily the numbers brought from the memory for arithmetic operations, and its presence is of no moment to the programmer. It is essential that the programmer be familiar with the roles played by the A and Q registers in the operations of arithmetic; many programming errors arise from placing operands in or removing results from the wrong register. The symbol R^3 has been taken over from the engineering description of the machine. In terms of this description, R_1 and R_2 correspond to A and Q respectively.

2.4 ADDITIONS (Order Type L). Before an addition instruction begins, the augend (i.e. the number which is to be added to) lies in the accumulator register A. During execution of the addition instruction, the addend is transferred from a specified memory location to the number register R^3. The digits of R^3 are then sensed through the complement gate unchanged, so that the addend (i.e. the number to be added) forms one of the inputs to the adder. The augend in A is the second adder input. The adder forms the sum which is transferred to the accumulator A, replacing the augend. The quotient register Q is undisturbed by the addition instruction.

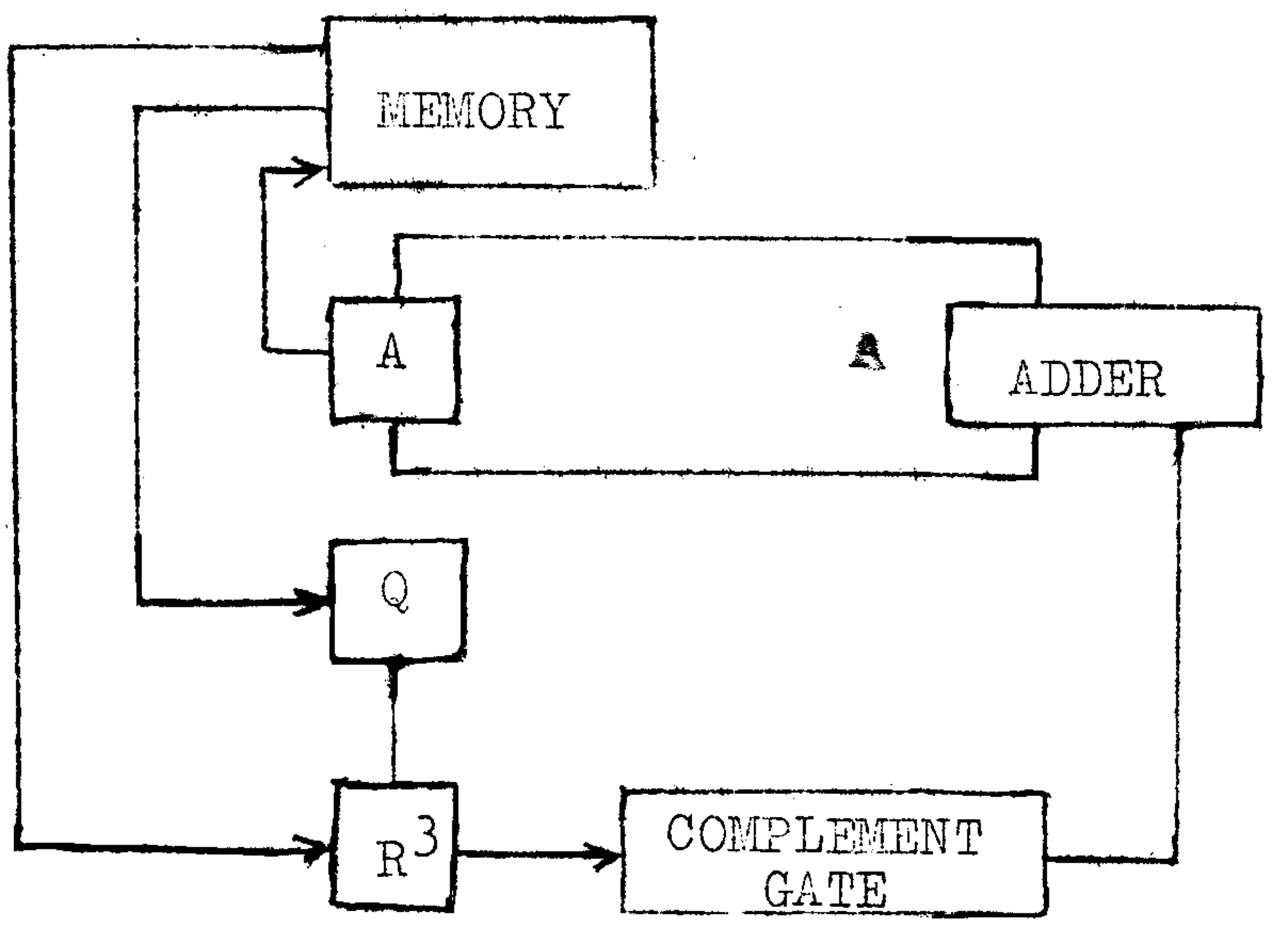

Figure 2.2
The SILLIAC Arithmetic Unit

Two variants of addition are "hold add" and "clear add". The "hold add" instruction leaves the augend in A undisturbed until the sum is formed by the adder.+ The "clear add" instruction is thus a transfer order which moves

+ The "clear add" instruction clears A to zero initially, thus setting the augend to zero.

a number from a specified memory location to the accumulator A.

2.5 SUBTRACTIONS (Order Type L). Subtraction in the SILLIAC arithmetic unit is performed by adding the complement of the subtrahend (i.e. the number to be subtracted) to the minuend (i.e. the number from which the subtrahend is to be subtracted). Before the subtract instruction begins, the minuend lies in the accumulator A, and is used as one of the two adder inputs. The subtrahend is brought from a specified memory location to R^3; its complement is formed by the complement gate and is used as the second adder input. The adder thus forms the difference by forming the sum of the minuend and the complement of the subtrahend. The adder output is then transferred to the accumulator A, replacing the minuend.

Either the "hold subtract" instruction or the "clear subtract" instruction can be used by the programmer. For the former instruction, the result of a previous operation is left in the accumulator A as minuend; for the "clear subtract", the minuend in A is set to zero, so that A contains the negative of the number in a specified memory location when the operation has ended.

2.6 ABSOLUTE VALUE ADDITION AND SUBTRACTION (Order Type L). It is possible to form the absolute value of the addend (or subtrahend) after it has been transferred from a specified memory location to R^3 and before it is added to the augend (or subtracted from the minuend) in the accumulator. Ordinarily the setting of the complement gate depends upon whether the instruction is an addition or a subtraction; for forming the absolute value of the number in R^3, it is necessary to sense the sign digit of R^3 as well. For example

if the addition of the absolute value of a negative addend is required, the sign digit of the addend in R^3 is sensed indicating that a complementation is necessary. The add instruction ordinarily does not require complementation; the net effect of the sensing of the add instruction and the R^3 sign digit is that the complement gate is set to form the complement of the addend.

2.7 INCREMENT ADD ORDERS (Order Type F). It was noted in Section 2.1 that the operation of complementation is performed by first forming the digitwise complement of the number held in R^3 and by then adding a unit in the least significant digital position of the adder. For the ordinary addition and subtraction orders (order type L), the least significant digit insertion occurs only when the complement gate is set to form the digitwise complement. In the SILLIAC, special increment add orders (type F) are provided. For these orders, the relationship between the setting of the complement gate and the insertion of the least significant digit is reversed. Thus, the "clear increment add" instruction sets the augend initially in A to zero and adds the addend from R^3 to 2^{-39} and places the result in A. Conversely, the "clear increment subtract" instruction sets the minuend in A to zero, replaces ones of the subtrahend by zeros and replaces zeros of the subtrahend by ones, and transfers the digitwise complement of the subtrahend thus formed from the adder to A. Detailed descriptions of further orders of this type are given on page 3-47.

2.8 ADD FROM Q (Order Type -) AND INCREMENT ADD FROM Q (Order Type +) INSTRUCTIONS. For the add from Q and increment add from Q instructions, the addend or subtrahend is transferred to the number register R^3 from the

Q register, rather than from a specified memory location. Otherwise, the + and - order types are the same as the corresponding F and L order types described in sections 2.4 through 2.7.

2.9 THE SHIFT ORDERS (Order Types 0,1). Since the right and left shift operations are fundamental in the SILLIAC arithmetic unit, specific shift instructions are provided for the programmer. These shift instructions are provided in several forms, viz.

(i) arithmetical shift (i.e. multiplication by $2^{\pm n}$) of the double length number contained in A and Q registers (see below),

(ii) arithmetical shift of A and Q registers separately,

(iii) logical shift of the contents of A and Q taken as an 80 digit number (i.e. a displacement of the 80 digits to the right or left), and

(iv) cyclic shift of the contents of A and Q taken as an 80 digit number (i.e. same as (iii) except that digits displaced beyond one end of the A or Q register appear at the other end of the A or Q register).

Shifts of the last three forms will be described in the next chapter. At this stage, the details of the arithmetical shift will be given in full as an introduction to the way in which a double length product is stored in SILLIAC.

In arithmetical shift of the contents of A and Q, all digits in both A and Q, except the sign digit q_0 of Q are shifted. A left shift (order type 0) of one digital

position replaces

$a_0, a_1, a_2, \ldots\ldots\ldots, a_{37}, a_{38}, a_{39}$ in A

and

$q_0, q_1, q_2, \ldots\ldots\ldots, q_{37}, q_{38}, q_{39}$ in Q

by

$a_1, a_2, a_3, \ldots\ldots\ldots, a_{38}, a_{39}, q_1$ in A

and

$q_0, q_2, q_3, \ldots\ldots\ldots, q_{38}, q_{39}, 0$ in Q.

The right shift (order type 1) of one digital position replaces

$a_0, a_1, a_2, \ldots\ldots\ldots, a_{37}, a_{38}, a_{39}$ in A

and

$q_0, q_1, q_2, \ldots\ldots\ldots, q_{37}, q_{38}, q_{39}$ in Q

by

$a_0, a_0, a_1, \ldots\ldots\ldots, a_{36}, a_{37}, a_{38}$ in A

and

$q_0, a_{39}, q_1, \ldots\ldots\ldots, q_{36}, q_{37}, q_{38}$ in Q.

From the behaviour of the left and right shift we see that we may consider the shifting to take place in a single register, written AQ, of 79 digits consisting of A followed by Q with q_0 deleted.

Thus, the left shift replaces

$a_0, a_1, \ldots\ldots\ldots, a_{38}, a_{39}; q_1, q_2, \ldots\ldots\ldots, q_{38}, q_{39}$

by

$a_1, a_2, \ldots\ldots\ldots, a_{39}, q_1; q_2, q_3, \ldots\ldots\ldots, q_{39}, 0$

while the right shift replaces

$a_0, a_1, \ldots\ldots\ldots, a_{38}, a_{39}; q_1, q_2, \ldots\ldots\ldots, q_{38}, q_{39}$

by

$a_0, a_0, \ldots\ldots\ldots, a_{37}, a_{38}; a_{39}, q_1, \ldots\ldots\ldots, q_{37}, q_{38}.$

The number n of shifts can be specified in the address digits of the shift order by the programmer. A left shift of n digital positions replaces [1] (AQ) (i.e. the contents of register AQ) by 2^n (AQ). Similarly, a right shift of n digital positions replaces (AQ) by 2^{-n} (AQ); n lies in the range $1 \leq n \leq 63$.

2.10 MULTIPLICATION (Order Type 7). Initially, the multiplier lies in the quotient register Q At the beginning of the multiplication instruction the multiplicand is transferred from a location specified by the address of the multiply order into the number register R^3, where it remains throughout the multiplication. The multiplication then consists of a sequence of additions and shifts. More precisely, a multiplier digit in q_{39} is sensed. There are two cases: (a) If $q_{39} = 1$, the multiplicand is added to the partial product in A. A right shift in AQ follows which halves the number in A and moves a new multiplier digit into q_{39} as well as transferring a product digit from a_{39} to q_1.

(b) If $q^{39} = 0$, only the right shift occurs, which transfers a_{39} to q_1 and transfers the next most significant digit of the multiplier into q_{39}. After 39 right shifts have occurred, the sign digit of the multiplier is sensed. If the multiplier is positive, the multiplication is complete, with a double precision product (sign digit and 78 non-sign digits) in AQ. For a negative multiplier x, the process of 39 shifts and conditional additions constitutes a multiplication of the multiplicand y by 1 + x (Equation 2.3) to form a product y(1 + x) = y + xy. In this case the SILLIAC automatically subtracts the multiplicand y to produce the correct product xy.

[1] The contents of a register the location of which is n is written as (n).

TYPE	INSTRUCTION	INITIAL CONDITIONS			FINAL CONDITIONS		NOTES
		Specified Memory Location	A Register	Q Register	A Register	Q Register	
L	Add	Addend	Augend	Arbitrary	Sum	Unchanged	Augend can be set to 0 or 1/2
	Subtract	Subtrahend	Minuend	Arbitrary	Difference	Unchanged	Minuend can be set to 0 or 1/2
7	Clear Multiply	Multi-plicand	Set to 0 at start of instruction	Multi-plier	Double Precision Product		q_0 set to 0
	Rounded Multiply	"	Set to 1/2 at start of instruction	"	Rounded Product	Usually Unwanted	"
	Hold Multiply	" y	Accumu-lant a	" x	$xy + 2^{-39}a$		"
6	Divide	Divisor	Double Precision Dividend		Residue	Rounded Quotient	
0	Left Shift (n . 1)	See Note 2	Double Precision Number x		$2^n x$		1. q_0 unchanged 2. Address digits specify number (n) of shifts.
1	Right Shift (n 1)	"	" x		$x/2^n$		

Table 2.1 Use of SILLIAC Registers During Arithmetic Instructions

Several variations of the basic multiplication procedure described in the previous paragraph are possible. The initial contents of A may be either 0, 1/2, or some quantity previously calculated. If we designate the initial contents of A by a, the final double precision product in AQ is $xy + 2^{-39} a$, where x and y are multiplier and multiplicand, respectively. For a = 0, the instruction specified is "clear multiply" and the result in AQ is the exact 78 digit signed product xy. For a = 1/2, the instruction specified is "roundoff multiply" and the result in A is a rounded 39 digit signed product. If a is arbitrary, the instruction specified is "hold multiply", and the result in AQ is $xy + 2^{-39}a$; the quantity a is thus added to the least significant part of the product in Q. In all multiplication instructions the sign digit q_0 of the quotient register is set to zero.

For each of the three types of multiplication instructions described in the previous paragraphs, four additional variations can be specified by the programmer. Either (n), - (n), |(n)|, or -|(n)| can be used as the multiplicand, where (n) is the number transferred from memory location n to the number register R^3.

2.11 DIVISION (Order Type 6). Initially, the double precision dividend lies in AQ. The divisor is transferred at the beginning of the division instruction from a memory location specified by the address of the divide order to the number register R^3. For positive divisor and dividend, the process is analogous to elementary long division. The divisor is subtracted from a partial remainder in A and the sign of the difference (in the adder) is sensed. If the difference is negative, 0 is inserted in q_{39} as quotient digit and AQ is doubled to form a new partial remainder. If

the difference is positive, 1 is inserted in q_{39} as quotient digit and the difference in the adder is placed in A, and (AQ) is doubled to form a new partial remainder. At each doubling of (AQ), q_1 is shifted into q_0 as well as into a_{39}. Thus after 39 steps q_0 has the sign of the quotient. The dividend is used as the initial partial remainder; after 39 quotient digits have been generated, the process is complete. A similar procedure is employed if divisor or dividend is negative or if both are negative.

The SILLIAC division has the following properties:

(1) A rounded quotient[1] Q is always generated. The roundoff is achieved by setting $q_{39} = 1$ in all cases.

(2) The 39th partial remainder is left in A and is called the residue r. The true remainder R corresponding to the rounded quotient is related to r approximately as follows:

$$R = r + (2q_0 - 1)\, y, \qquad (2.4)$$

where y is the divisor and q_0 is the sign digit of the quotient. Thus, if the quotient is positive

$$R = r - y,$$

and if the quotient is negative

$$R = r + y$$

The equations given above for the true remainder are valid if the absolute value of the quotient is less than one, and they yield results in error by not more than 2^{-39}.

---

1. To simplify later formulae, the contents of Q are written in this section as Q.

(3) The sign digit of the quotient replaces the least significant (78th) digit of the double-precision dividend. One effect is that the least significant digit of the residue (a_{39}) is the same as the sign digit of the quotient q_0.

(4) If we have <u>a priori</u> knowledge of the true value q of the quotient (such as, for example, a division of qy by y), the relationship between the machine quotient Q and the true quotient is

$$Q = q + 2^{-39}(1 - q_{39})(1 - 2y_0) \qquad (2.7)$$

where q_{39} is the least significant digit of q and y_0 is the sign digit of the divisor y. Equation 2.7 is essentially a description of the division roundoff. If $q_{39} = 1$, then $Q = q$ and the machine quotient is the true quotient. If $q_{39} = 0$ the nature of the roundoff depends upon the sign of the divisor. Suppose, for example, that q is 0.101; then Q is either 0.101000...001 or 0.100111...111 depending upon whether the divisor was positive or negative, respectively.

2.12 PRECISE CALCULATION OF THE DIVISION REMAINDER.*

We define the remainder R in relation to a quotient Q, a divisor y, and a dividend d by the equation

$$Qy + 2^{-39} R = d,$$

or

$$R = 2^{39} (d - Qy). \qquad (2.8)$$

However, the exact relationship involving the SILLIAC

residue r and including the replacement of the least significant dividend digit d_{78} by the quotient digit q_0 is

$$Qy + 2^{-39}\left[r + (2q_0 - 1)y\right] = d + 2^{-78}(q_0 - d_{78}). \quad (2.9)$$

Solving for R, we have

$$R = 2^{39}(d - Qy) = r + (2q_0 - 1)y + 2^{-39}(d_{78} - q_0). \quad (2.10)$$

Equation 2.10 gives the exact expression for the remainder R which corresponds to the machine quotient Q.

If the remainder $\overline{R}$ corresponding to the true quotient q is desired, then $\overline{R}$ is defined as

$$\overline{R} = 2^{39}\,(d - qy) \quad (2.11)$$

The value of $\overline{R}$ is found by substituting equation 2.7 in equation 2.9 and solving for $\overline{R}$, which yields

$$\overline{R} = r + \left[\overline{2q_0 - q_{39} - (1 - q_{39})2y_0}\right] y + 2^{-39}(d_{78} - q_0). \quad \ldots\ldots (2.12)$$

2.13 THE DIVISION HANGUP. SPECIAL CASES OF DIVISION. Circuits are incorporated in the SILLIAC for stopping the SILLIAC if the quotient resulting from a division exceeds one in absolute value. The sign digit of the quotient is predictable from the signs of divisor and dividend. The sign digit of the quotient is calculated in the SILLIAC by comparing the dividend and divisor arithmetically. Thus, by sensing the sign digits of quotient, divisor, and dividend, it is possible to detect the fact that the quotient exceeds one in absolute value, and stop the SILLIAC.

The equations derived in Sections 2.10 and 2.11 are valid only if the absolute value of the dividend is less than the absolute value of the divisor. When absolute

values of dividend and divisor are equal, the SILLIAC generates a quotient $-1 + 2^{-39}$ if the dividend is negative and the divisor is positive. The quotient is $+1 - 2^{-39}$ if dividend and divisor are both negative. If the dividend is positive and equal to the absolute value of the divisor, the SILLIAC will stop. If the divisor is -1, the SILLIAC generates a quotient which is the digitwise complement of the dividend, except for the quotient roundoff.

2.14 MEMORY TO Q (Order Type 5), STORE FROM A (Order Type 4) AND STORE FROM Q (Order Type N) INSTRUCTIONS. Instructions are provided in the SILLIAC for transferring a number from a specified memory location to the Q register, and for transferring a number in A or Q to a specified memory location. The former instruction can be used to transfer the multiplier to Q before a multiplication; the latter instructions are used to transfer a result from A or Q to the memory.

2.15 ILLUSTRATIVE EXAMPLES.*

A. The Leapfrog I Division Test. In the Leapfrog I division test[1] the product xy is formed and is then divided by y. The sum of the quotient in Q and residue in A is formed and stored. The sum of quotient and residue is then calculated independently and compared with the sum previously stored.

The value of the machine quotient Q in the quotient register after division of xy by y is given by equation 2.7 with q = x. Thus

$$Q = x + 2^{-39}(1 - x_{39})(1 - 2y_0).$$

The value of the residue r left in the accumulator is found from equation 2.12 by setting $\overline{R} = 0$, x = q, and solving for r:

1. This is an Engineering test routine.

$$r = \left[x_{39} + (1 - x_{39})2y_0 - 2x_0\right] y + 2^{-39}(x_0 - x_{39}y_{39}).$$

The sum Q + r, after rearranging terms, is

$$Q + r = x - 2^{-39}x_{39}y_{39} - x_0(y - 2^{-39}) + (1 - x_0)y +$$
$$(1 - x_{39})(2y_0 - 1)(y - 2^{-39}).$$

The independent calculation of Q + r thus consists of forming $x - 2^{-39}x_{39}y_{39}$ and either adding y or $(-y + 2^{-39})$ depending upon whether x is positive or negative; and finally, if x_{39} is 0, adding or subtracting $(y - 2^{-39})$ depending upon the sign of y.

B. The "Double Precision" Division. It is sometimes convenient to consider a single precision divisor y as exact and form a "double precision" quotient $s + 2^{-39}t$ utilizing the double precision dividend d originally in AQ. The procedure used is as follows:

1. Form d/y, yielding $s + 2^{-39}$ in Q and $r + 2^{-39}s_0$ in A.
2. Shift right, forming r/2 in A and s/2 in Q.
3. Form r/2 + y leaving T in Q and a residue u in A.
4. Assemble $s + 2^{-39}t$ by setting t_0 (the sign digit of T) to zero, inserting s/2 in A and shifting left once.

The precision of $s + 2^{-39}t$ can then be calculated as follows:

Step 1 yields s and r such that (by equation 2.9)

$$(s + 2^{-39})y + 2^{-39}\left[r + 2^{-39}\, s_0 + (2s_0 - 1)y\right] = d + 2^{-78}(s_0 - d_{78}),$$

or

$$sy + 2^{-39}(r + 2s_0y) = d - 2^{-78}\, d_{78}. \qquad (2.13)$$

Step 3 yields

$$Ty + 2^{-39}\left[u + (2t_0 - 1)y\right] = r/2 + 2^{-78}t_0 \qquad (2.14)$$

The following substitutions are made:

(a) $s_0 = t_0$, since the sign of r/2 is the same as the sign of d.

(b) From step 4, $t = 2(T + s_0)$ or $T = 1/2t - s_0$.

Substituting in equation 2.14 and solving for r, we have

$$r = (t - 2s_0)y + 2^{-38}\left|u + (2s_0 - 1)y\right| - 2^{-77}s_0.$$

Substitution of the value for r in equation 2.13 yields

$$sy + 2^{-39}\left[(t - 2s_0)y + 2s_0y\right] + 2^{-77}\left[u + (2s_0 - 1)y\right] - 2^{-116}s_0 = d - 2^{-78}d_{78}.$$

We thus have

$$(s + 2^{-39}t)y + 2^{-78}\left[2u + 2(2s_0 - 1)y + d_{78} - 2^{-38}s_0\right] = d.$$

It can be shown that $u + (2s_0 - 1)y < 1$ so that the quantity within square brackets is less than 3 in absolute value. We conclude that the quotient $s + 2^{-39}t$ is in error by not more than 2^{-76}.

The program for forming a "double precision" quotient from a double precision dividend and an exact single precision divisor is given in words 57 to 66 of library routine A4 entitled, "1.7 Precision Floating Decimal". A major difficulty encountered is the formation of r/2 from r (Step 2 above), for r may be as large as 2y and may therefor exceed range. It is therefore necessary to set the sign digit of r/2 to that of the original dividend d.

2.16 INTEGER OPERATIONS. It is sometimes desirable

to use integers for computations in the SILLIAC. Suppose we have an integer a stored in the memory or arithmetic unit. In terms of the formulation of previous sections, we would store $2^{-39}a$, where a lies in the range $-2^{39} \leq a < 2^{39}$. If we wish to add or subtract two integers a and b, no difficulty is encountered, for $2^{-39}a \pm 2^{-39}b = 2^{-39}(a \pm b)$, indicating that the correct sum or difference lies in A after the instruction is performed. Multiplication of two integers a, b yields $(2^{-39}a)(2^{-39}b) = 2^{-78}$ ab. The product ab lies in AQ and is in the range $-2^{78} \leq ab < 2^{78}$. If the programmer scales all quantities so that the product remains in the range $-2^{39} \leq ab < 2^{39}$, then the 40 digit signed product can be transferred to A by a left shift of 39 digital positions. It should be noted that the sign digit of Q is set to zero during the multiplication so that for positive products, (Q) = ab, if $0 \leq ab < 2^{39}$.

Division of integers presents certain difficulties. An example is given here of a method of dividing a positive dividend a by a positive divisor b to yield a quotient f and remainder g.

The steps are as follows:

(1) Place the dividend $2^{-78}a$ in AQ. $0 \leq a < 2^{77}$,

(2) shift left one digital position, leaving 2a in AQ with $q_{39} = 0$,

(3) divide by $2^{-39}b$, leaving $(2f + 1)2^{-39}$ in Q and $(2g)2^{-39}$ in A. $0 \leq b < 2^{-39}$,

(4) shift right one digital position, leaving $2^{-39}f$ in Q and $2^{-39}g$ in A.

It can be proved that bf + g = a by substitution of the appropriate quantities in equation 2.9, as follows:

$$(2f + 1)2^{-39}(2^{-39}b) + 2^{-39}\left[2^{-39}(2g) - 2^{-39}b\right] = 2^{-78}(2a)$$

which yields

$$2^{-78}(2bf + b + 2g - b) = 2^{-78}(2a) \text{ or } bf + g = a.$$

The ranges of f and g are $0 \leq f < 2^{38}$ and $0 \leq g < 2^{38}$.

2.17 SUMMARY. In the SILLIAC arithmetic unit are two registers, A and Q, which are directly accessible to the programmer. A single arithmetic order of a program utilizes the initial numerical operands in A, in Q, and in a specified memory location, and transforms these quantities to produce desired results which are left in the registers. The programmer must know where the operands are initially located and where the results are to be found. The functions of the registers for the operations of arithmetic are indicated in Table 2.1.

The SILLIAC has a fixed point arithmetic unit; the binary point is fixed so that any number x used in computation must lie in the range $-1 \leq x < 1$. The programmer must ensure that all quantities remain within this range during a computation. The sign of a numerical quantity is indicated by the leftmost of the 40 binary digits stored in a register or a memory location. The sign digit is 0 for a positive number, 1 for a negative number.

CHAPTER 3

THE ORDER CODE

In carrying out a complete calculation, the SILLIAC extracts from its high-speed store and executes one at a time a sequence of orders, each of which calls for one or another of the elementary operations that the machine can perform. Before they can be so used the orders must be entered into the machine's main high-speed store. For this purpose the orders are coded in the manner set out in this chapter. The machine is designed so that any storage location in the memory may be used either for orders or for numbers, the only distinction being that the control must be instructed properly so that orders and numbers will be treated appropriately.

Already in Chapter 2 the various operations that SILLIAC can carry out upon numbers have been briefly described. These operations are not in themselves sufficient for the make up of a complete practical program for a given calculation. In addition the following types of operation are provided:

(a) operations for the input of information from punched paper tape, and for the output of information on to punched paper tape;

(b) transfer-of-control operations, for controlling the sequence in which the various parts of a complete program are carried out;

(c) address-modification operations for assisting in the organisation of repetitive loops within a program;

(d) null orders;

(e) stop orders.

Explanatory notes on these several types of operation are included in this chapter. In addition the SILLIAC's

complete repertoire includes a set of operations associated with the use of its magnetic tape backing store; for an explanation of these operations the reader is referred to Chapter 12.

In the final pages of this chapter there is given first the complete list of all possible orders that can be used, together with the operations that they specify, and finally an abbreviated reference list of the more commonly used orders.

3.1 THE MAKEUP OF ORDERS. An order for a digital computer consists, in general, of a function part to say what to do, and one or more address parts to say where to find the quantities to be used in carrying out the operation, and where to put the result. In general any arithmetical operation has two operands and a result. If all of these were to be located in the store, three address parts might be used in a single order to specify the locations. A machine using such a code would be said to have a three-address order code. The SILLIAC however would carry out such an operation in three separate orders, each one having one address only associated with it. Such a code is known as a one-address order code. The single address part in an order specifying an arithmetical operation for the SILLIAC is used to locate one of the operands; the other operand and the result are always to be found in one or the other of the two registers (A and Q) provided in the arithmetic unit of the machine.

In a SILLIAC order eight binary digits (or "bits") are used for the function part, and since the main store has 1024 locations, a further ten binary digits ($1024 = 2^{10}$) are required for the address part. The function digits are placed on the left, and the address digits on the right; in

between are a further two digits which are at present unused[1] but which would be used should the size of the store ever be increased to 4096 ($=2^{12}$) locations. A complete order is thus spread over 20 digits, and accordingly two such orders, called an order pair, are packed into one 40-digit storage location in the manner shown in Fig. 3.1.

8-DIGIT FUNCTION	10-DIGIT ADDRESS	8-DIGIT FUNCTION	10-DIGIT ADDRESS
LEFT-HAND ORDER DIGITS 0-19		RIGHT-HAND ORDER DIGITS 20-39	

Figure 3.1
Order Pair Makeup

The eight function digits of an order, for convenience in writing, are split into two groups of four, and each group of four is represented by a single base-16 (sexadecimal) digit. Thus each function is coded as two sexadecimal digits. The sexadecimal number system as used with SILLIAC, uses the decimal digits 0, 1,...,9 for the first ten digits, and the symbols + (or K), - (or S), N,J,F,L for 10,11,12,13,14,15.

As an example of an order pair, let us consider the following 40 binary digits:

1111010100000011101010000000000000110

When divided into function and address digits, these digits look like this:

---

1 Non-zero digits in these positions will have no effect at present. However, if the store is extended, this will no longer be true.

LEFT-HAND ORDER		RIGHT-HAND ORDER	
1111 0101 00	0000011101	0100 0000 00	0000000110
FUNCTION	ADDRESS	FUNCTION	ADDRESS

The left hand function is made up of the two 4-digit numbers 1111 0101 which are the sexadecimal digits L5. The left-hand address is interpreted as an integer which may range from 0 to 1023 in decimal notation, or from 0 to 3LL in sexadecimal notation.

In sexadecimal notation the left-hand address is 1J which corresponds to the decimal number 29. Thus the left hand order is L501J where the 0 has been supplied so that all 20 binary digits (including the two unused digits) are accounted for. (We could have set the unused digits to 1's and used N rather than 0 if we had wished.)

Similarly the right-hand order is 40006, and we have, in sexadecimal notation, the order pair

L501J 40006.

This order pair says: "Copy the contents of storage location 01J into the accumulator; store the accumulator contents at storage location 006."

It would be quite inconvenient if one always had to write addresses in sexadecimal form, and, in fact, this is not necessary if there is first placed in the machine a set of orders (a "routine") which will cause it to read program tape with addresses in decimal form, and convert them to sexadecimal (i.e. binary) form for use inside the machine. Routines of this type are available for the SILLIAC. The simplest of these are called Decimal Order Input Routines. The use of one of these is explained in Chapter 5.

3.2 INPUT AND OUTPUT ORDERS. On the paper tape used as an input and output medium for SILLIAC, a punched hole

is used to represent the binary digit 1, and the absence of a hole the binary digit 0. There is space across the tape for five punching positions, and therefore each row of holes or "character" on the tape can represent any number from 00000 to 11111, or 0 to 31 in decimal notation. For full details of the tape code the reader is referred to Chapter 9. Here it is sufficient to point out that for the representation of numerical information, whether in decimal or sexadecimal form, four punching positions are sufficient. Accordingly there is provided in the SILLIAC one set of input and output orders which are concerned solely with 4-digit characters (i.e. sexadecimal characters, referred to as 4-level characters) having no hole punched in the fifth position on the tape. These are orders of type 8.

In order to input or output 5-digit characters, which may include a hole in the fifth position, a second set of input-output orders is provided. This second set of orders is used in general for handling non-numerical information including, in particular, teleprinter control characters such as space, line-feed, carriage-return etc. These are orders of type 9.

An output order of type 8 can punch only 4-level characters on the output tape - it can never punch a hole in the fifth position. An input order of type 8 will read only 4-level characters from the input tape -- any character with a hole punched in the fifth position is ignored. An output order of type 9 can punch 4- and 5-level characters, the actual character punched depending upon the detailed specification of the order. Similarly an input order of type 9 can read 4- or 5-level characters from the input tape.

It is frequently desirable to input or output more than one tape character by means of a single SILLIAC order. This

has been made possible by making use of the operation of shifting in the A and Q registers of the machine. (See Chapter 2.) It should be noted that the shifting which occurs during input and output orders is always of the non-arithmetic variety (i.e. all 80 digits of A and Q are shifted, with no special treatment of the sign digits); further logical shifting is used for input orders, and cyclic shifting for output orders. The shifting is to the left in the first (numerical) type of input and output mentioned above, and to the right in the second (5-level) type of input and output.

Input orders of both types input information into the arithmetic unit; output orders of type 8 output information from the arithmetic unit, but the characters output by orders of type 9 do not come from the arithmetic unit - instead they are completely specified by the address part of the order itself.

3.3 EXECUTION OF ORDERS. In the course of its operation the SILLIAC reads out orders from the store and places them a pair at a time, ready for execution, in the ORDER REGISTER. The content of the order register is displayed on lights across the bottom of the machine.

The program is begun with a particular order chosen by the programmer. Let us suppose that it is the left-hand order at location 10. (We shall refer to addresses in decimal notation.) The machine's control will put into the order register the order pair from location 10. Then, until it encounters an order which varies the sequencing, the machine will follow a fixed pattern in executing orders. It will carry out the left-hand, and then the right-hand order of the pair in the order register. Then it will read the

order pair from location 11 into the order register and again execute the left- and right-hand orders. It will continue to read out and execute order pairs from successive storage locations until one of two things occurs:

(a) one of the orders brought out stops the machine,
or (b) one of the orders brought out says "bring the next order pair from storage location so-and-so".

The second kind of order (b) is called a <u>control transfer order</u>. It permits the programmer to change the sequencing of orders and provides the flexibility required for iterative processes. It works in the following way.

Let us suppose that after the machine has executed the left-hand order at location 17, the programmer wishes to move to a sequence of orders beginning, say, with the right-hand order at location 35. Then the right-hand order at location 17 will say "Transfer control to the right-hand order at location 35". The execution of this order will consist of arranging that the next order pair be brought from location 35, and that the left-hand order be skipped. Having carried out the right-hand order in the order register, the control will bring out the next order pair from location 36 and proceed in the usual way.

A control transfer order may be itself either a left-hand or a right-hand order. If it is the former the right-hand member of its pair will be skipped out, and the machine will proceed with either the left- or right-hand order of the new order pair, as determined by the control transfer order.

Control transfer orders may be either <u>unconditional transfer orders, or conditional transfer orders</u>. The unconditional kind (orders of type 2) have just been described.

The conditional transfer orders do the same thing provided that a certain condition is satisfied in the machine. If the condition is not satisfied the machine ignores the control transfer order and goes straight on to the next order in sequence.

In the SILLIAC there are two sets of conditional transfer orders, and each set has its own condition for determining whether the transfer should be performed or ignored. In the first set (orders 30, 32, 34, 36) the condition is the sign of the number in the accumulator. If this number is positive or zero (i.e. if the digit a_0 is a 0) the transfer will be performed, but if negative (i.e. if a_0 is a 1) the transfer will be ignored. In the example used above, if the right-hand order at location 17 had been a conditional order of this type, then if the accumulator had held zero or a positive number the next order executed would have been the right-hand order at location 35. If, however, the accumulator had held a negative number, the next order executed would have been the left hand order at location 18.

In the other set of conditional orders (31,33,35,37) the state of the overflow indicator determines whether the control transfer is to be performed or not. The machine contains an indicator which is set to indicate overflow[1] whenever overflow occurs. This may be in addition or subtraction, or in left-shift or in one unusual case of multiplication, namely that in which -1 is multiplied by -1 with the accumulator initially positive or zero. The indicator is cleared to

---

[1] Overflow is said to occur if the result of an operation, x, is such that $x \geq 1$ or $x < -1$.

cancel any indication of overflow by any order which involves clearing the accumulator to zero, and by any order which involves right shifting (that is to say right shift, multiplication and input, output and playback orders of type 9). An order calling for transfer of control conditionally upon no overflow will be ignored if the indicator is set to indicate overflow.

A further facility provided in connection with control transfer orders is that, if desired, the machine may be caused to stop before executing any transfer of control. The machine may then be restarted manually by means of a switch on the control panel. For the provision of this stop facility every controltransfer order has two forms. One form of each order will cause the machine to proceed without interruption in the manner described above, while the other form of the order can cause a stop. A conditional control transfer order of the stopping type causes a stop only when the condition for transfer of control is satisfied. The stop facility may be disabled by setting a switch on the control panel to a position labelled IGNORE. When the switch is on this position the two forms of order just described become identical - neither causes a stop.

The location in the store from which the next order pair will be brought is determined by a counter located in the control unit of the machine, and called the ORDER COUNTER. The address in this counter is advanced by one immediately after each order pair is brought out into the order register. The manner in which a control transfer is produced during the execution of a control transfer order, is thus simply by changing the contents of the order counter. These contents are displayed on lights, in sexadecimal form, on a panel just above the control panel of the machine.

3.4 MODIFICATION OF ADDRESSES. A common feature of computer programs is that some orders in the program have to be executed a number of times, but with a different address each time. For example, the address may require to be increased by one each time the order is executed so that the order may refer in succession to the several numbers of a list stored in successive storage locations. This situation can be met, without special facilities, simply by bringing the order concerned out into the arithmetic unit, adding one to its address, and writing it back into the store ready for the next time it is to be used. This procedure is, however, rather clumsy if, as usual, there are in a section of a program several orders all requiring similar alteration of their addresses. Most computers include some special facility for simplifying the procedure in these cases.

In the SILLIAC this special facility for modifying addresses takes the form of two special orders called indexing orders 3- and 3L. The effect of these orders is to cause an index number to be added to (3L) or subtracted from (3-) the address of the next following order; this change to the address is made only in the control section of the machine without changing the order in the store. The address of the indexing order specifies the location of the index number, and the index number itself is the number contained in the right hand address position of the specified store location.

Suppose for example that storage location 10 contains the number 15 in its right-hand address part, and then

consider the two successive orders

3L 10
L4 25

These might be members of the same pair, or, equally well, might be the right-hand member of one pair followed by the left-hand member of the next. By itself the order L4 25 would cause the contents of location 25 to be added into the accumulator; but when preceded by 3L 10, it is the contents of storage location 40 (=25+15) that will be added into the accumulator, since location 10 contains 15 in the right-hand address part. It should be noted that the addition of the numbers 25 and 15 occurs only transiently during the execution of the order. The order L4 25, as it stands in the store of the machine, is left quite unaltered.

The addition, or subtraction, of an index number to an order by an indexing order affects only the address part of the order; it cannot alter the function digits. Thus, if the index number 100 is subtracted from the order L4 50 by an indexing order the result will be interpreted as L4 974 for

$$974 = 50 - 100 + 1024$$
$$\equiv |50 - 100| \text{ Modulo } 2^{10}.$$

Indexing orders may be used to modify the addresses of all orders except those mentioned in the next paragraph. For example they may be used to vary the effective address of a control transfer order and to vary the number of shifts in a shift order or input/output order of type 8. An indexing order may even be used to vary the address of another indexing order.

An important restriction on the use of indexing orders is that in no circumstances should one be used to precede a

magnetic tape order. To do so will cause malfunctioning of the magnetic tape equipment. Another exceptional case occurs in connection with output orders of type 9; for details the reader is referred to the detailed description of indexing orders later in this chapter.

3.5 NULL ORDERS The grouping of SILLIAC orders into pairs sometimes causes minor inconvenience to the programmer. Some aspects of the available orders, and certain conventions used in programming, will sometimes make it necessary that certain orders should appear as left-hand orders rather than right-hand, or vice versa. To fulfil these requirements it will sometimes be necessary to use a waste order--one that simply fills in a gap without doing any harm. To have available, for use in such cases a special null order --one which simply sends the SILLIAC on to the next order without any other action--is hardly a necessity but it is a convenience, and such orders have been provided in SILLIAC.

The SILLIAC orders 39 and 3J are both null orders and may be used whenever required in a program except in between an indexing order and the order whose address it is intended to modify. An expert programmer may use one of them even there if he attends carefully to the details of these orders as set out later in this chapter.

3.6 STOP ORDERS To be able to stop the SILLIAC at the end of a calculation is of course very necessary, and there is no lack of orders for doing this. Mainly for engineering reasons, all orders the second sexadecimal function digit of which is 8, +, N, or F, will stop the machine. Out of this total of 64 stop orders, one, namely OF is that normally used to stop the machine at the end of a calculation.

Orders with second function digit 8, +, N, or F stop

SILLIAC without having any effect on the content of the arithmetic register or the memory. After being stopped by one of these orders SILLIAC can be restarted by operating the white switch on the control panel, and when restarted proceeds with the orders following the stop order.

3.7 ORDER TYPES AND VARIANTS. With two sexadecimal function digits, the total number of code combinations for representing orders is 16 x 16 = 256. However, as mentioned above, 64 of these merely stop the machine, and of the remainder only about 150 represent distinct, usable orders. This figure of 150 contains many that are but rarely used, and the general programmer can manage with a much smaller vocabulary. For reference purposes it is necessary that this manual should contain the complete list and this is given in the following pages.

The two sexadecimal function digits of an order give the order type and the order variant. We shall refer to them as T- and V-digits, respectively. In the example L4 cited earlier the T-digit is L which denotes addition; the V-digit is 4 which denotes one of the variants of the addition type of order. The order types are given in Table 3.1. In orders of each type the address digits have a particular meaning; this is set out in Table 3.2.

T-Digit	Order Type
0	Left Shift
1	Right Shift
2	Unconditional Control Transfer
3	(Conditional Control Transfer
	(Address Modification
	(Null orders
4	(A) to Store
5	Store to Q
6	Divide
7	Multiply
8	(Magnetic Tape
	(Input/Output
9	(Magnetic Tape
	(5-level Input/Output
+	Increment add from Q
-	Add from Q
N	(Q)to Store
J	Collate in Q
F	Increment Add
L	Add

Table 3.1
Order Types

ORDER TYPE		MEANING OF ADDRESS
T-V-Digits	Operation	
0,1	Shift	Number of shifts (interpreted modulo 64).
2,3	Control Transfer	Location of next order pair.
3-, 3L	Indexing	Location of index number.
39, 3J	Null	No significance.
4	A to Store	Location at which storage will occur.
5	Store to Q	Location of word read into Q.
6	Divide	Location of divisor.
7	Multiply	Location of multiplicand.
80, 82	Input/Output	Determines number of characters input or output, and number of shifts (interpreted modulo 64).
84	Record	Engineering significance only - must be 5.
86	Rewind (full word order)	L.H. address determines which magnetic tape will be rewound. R.H. address has no significance.
8L	Record Block No. (full word order)	L.H. address determines which tape and which direction of motion, R.H. address (12 digits) contains Block No. to be recorded.
91	5-level input	Determines no. of characters read (interpreted modulo 64).
92	5-level output	Determines character punched and no. of times it is to be punched.
94	Playback	Determines which magnetic tape and which direction of motion.
96	Search (full word order)	L.H. address determines which tape and which direction of motion. R.H. address (12 digits) contains Block No. to be searched for.
+, -	Add from Q	No significance.
N	Q to Store	Location at which storage will occur.
J	Collate in Q	Location of word to be collated with (Q).
F, L	Add	Location of addend.

Table 3.2
Meaning of Address Digits

Let us now consider the variants obtained by selecting various values of the V-digit. The sexadecimal V-digit is made up of four binary digits V8, V4, V2, and V1 and there is a degree of regularity in the significance of these digits which helps considerably in remembering the various orders.

V1 and V8 Digits

(a) In all orders except type 3 and magnetic tape control orders, the condition V1 = 1 will cause the accumulator to be cleared to zero before the operation specified by the order is carried out; if V1 = 0 A will not be cleared. Thus an odd V-digit means that A will be cleared first.

(b) With the same exceptions as above, the condition V1 = 1, V8 = 1 will cause the accumulator to be first cleared and then set to contain 1/2 at the start of the order. Thus an odd V-digit greater than 8 means that 1/2 is placed in A first.

(c) Without any exceptions the condition V1 = 0, V8 = 1 (i.e. V = 8, +, N, or F) will cause the machine to stop.

(d) In type 3 orders V8 = 0 gives conditional transfer of control orders; V8 = 1 gives indexing and null orders. The determining condition in conditional transfers is the accumulator sign if V1 = 0, and the overflow indicator if V1 = 1.

V2 and V4 Digits

The V2 and V4 digits affect orders in the manner set out in Table 3.3.

TYPE	V4	EFFECT OF V4	V2	EFFECT OF V2
0,1	0	Arithmetical Shift	0 1	(AQ) as one long number (A) and (Q)separately
	1	Logical Shift	0 1	Noncyclic Cyclic
2,30 to 37	0 1	Transfer to right-hand order Transfer to left-hand order	0 1	Stop if switch on OBEY Do not stop
38 to 3L	0 1	Preserve indexing Cancel indexing	0	Null order
	0 1	Subtract index Add index	1	Indexing order
4,N	0 1	No effect	0	Store full word
	0 1	Store right-hand address Store left-hand address	1	Store address only
5,J		No effect		No effect
6	1	Must be 1	1	Must be 1
8,9	0	Input/Output	0 1	Input Output
	1	Magnetic Tape	0 1	Playback or record (half word orders) Tape control operations(full word 0 orders)
7,+,-, F,L	0 1	Subtract Add	0 1	Number Absolute Value

In the detailed statement of the order code, which now follows, the following notation is used:

A	The accumulator register
Q	The multiplier-quotient register.
a_r, q_r	$0 \leq r \leq 39$, digits of A and Q; for the sign digit $r = 0$.
AQ	The 79 binary digit double register formed from A and Q by omitting q_0.
(R)	Contents of register R.
(n)	Contents of storage location n.

When it is desired to refer to the contents of a register R both before and after a given operation, the initial content will be written (R) and the final content (R)'.

It should be noted that the statement which follows is intended to be definitive. That is to say, relevant side effects are stated for completeness: these are not always relevant to the main purpose of the orders, but may occur as a by-product of the engineering design of the machine. If they are regarded as harmless, no action has been taken to prevent them; however, it is essential that the programmer should know of their existence.

Left Shift

OVn

The number n is interpreted modulo 64; if so interpreted, n = 0, the machine will stop; if not, carry out n times in each case the shift operation described below, for n = 1.

Values of V	Variant
0	Arithmetical left shift of (AQ); that is to say, if n = 1, replace the initial contents $a_0a_1\cdots a_{38}a_{39} \mid q_0q_1\cdots q_{38}q_{39}$ of A and Q by: $a_1a_2\cdots a_{39}q_1 \mid q_0q_2\cdots q_{39}0$.
1	Clear A, then same as 0.
2	Arithmetical left shift of (A) and (Q) separately; that is to say, if n = 1 replace the initial contents of A andQ by: $a_1a_2\cdots a_{39}0 \mid q_1q_2\cdots q_{39}0$.
3	Clear A, then same as 2.
4	Logical left shift of the 80 digits in A and Q: that is to say, if n = 1 replace the initial contents of A and Q by: $a_1a_2\cdots a_{39}q_0 \mid q_1q_2\cdots q_{39}0$
5	Clear A, then same as 4.
6	Cyclic left shift of the 80 digits in A and Q: that is to say, if n = 1, replace the initial contents of A and Q by: $a_1a_2\cdots a_{39}q_0 \mid q_1q_2\cdots q_{39}a_0$
7	Clear A, then same as 6.
8,+,N,F	Stop,
9,-,J,L	Make (A) = 1/2, then same as 0,2,4,6,respectively.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, simultaneously with setting (A) = 0 or 1/2.

(2) If at any time during the execution of any left shift order, the digit a_0 changes value, the overflow indicator will be set.

(3) The order OF is conventionally used for the final stop at the end of a program.

Right Shift

1Vn

The number n is interpreted modulo 64; if, so interpreted, n = 0, the machine will stop; if not, carry out n times in each case, the shift operation described below, for n = 1.

Values of V	Variant
0	Arithmetical right shift of (AQ): that is to say, if n = 1 replace the initial contents $a_0a_1\cdots a_{38}a_{39} \mid q_0q_1\cdots q_{38}q_{39}$ of A and Q by: $a_0a_0a_1\cdots a_{37}a_{38} \mid q_0a_{39}q_1q_2\cdots q_{37}q_{38}$
1	Clear A, then same as 0.
2	Arithmetical right shift of (A) and (Q) separately; that is to say, if n = 1 replace the initial contents of A and Q by: $a_0a_0a_1\cdots a_{37}a_{38} \mid q_0q_0q_1\cdots q_{37}q_{38}$.
3	Clear A, then same as 2.
4	Logical right shift of the 80 digits in A and Q: that is to say, if n = 1 replace the initial contents of A and Q by: $0a_0a_1\cdots a_{37}a_{38} \mid a_{39}q_0q_1\cdots q_{37}q_{38}$.
5	Clear A, then same as 4.
6	Cyclic right shift of the 80 digits in A and Q: that is to say, if n = 1 replace the initial contents of A and Q by: $q_{39}a_0a_1\cdots a_{37}a_{38} \mid a_{39}q_0q_1\cdots q_{37}q_{38}$.
7	Clear 1, then as 6.
8,+,N,F	Stop.
9,-,J,L	Make (A) = 1/2, then same as 0,2,4,6, respectively.

NOTES

(1) All right shift orders will reset the overflow indicator so as to cancel the indication of any previous overflow.

(2) For convenience, 10n can be said to operate on (AQ) (i.e. q_0 is unaffected), 12n on (A) and (Q) and 14n and 16n on (A/Q).

These three abbreviations serve to describe the treatment of q_0 and the "splitting"of the A and Q registers.

Unconditional Control Transfer

2Vn

Bring the next order pair from storage location n and start with the left-hand or right-hand order, stopping beforehand or not, according to the value of V.

Values of V	Variant
0	Stop. Upon starting with the blackswitch the first order performed will be the right-hand order at location n. If the black switch is set at IGNORE, the machine will proceed with the order without stopping.
2	Transfer control to the right-hand order at location n.
4	Same as 0, except take left-hand order.
6	Same as 2, except take left-hand order.
1,3,5,7	Clear A, then same as 0,2,4,6 respectively.
8,+,N,F	Stop.
9,-,J,L	Make (A) = 1/2, then same as 0,2,4,6 respectively.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, simultaneously with setting (A) = 0 or 1/2.

(2) If, after the machine has stopped on a stop-transfer-of-control order, it is started with the white instead of the black switch, the transfer of control is not performed. Instead, the machine proceeds with the next order in sequence, or if the

stop-transfer-of-control order is a right-hand order transferring control to a right-hand order, the machine proceeds with the right-hand order of the next order pair in numerical sequence.

Values of V	Variant
0,2,4,6	If (A) $\geq$ 0 proceed as in the corresponding 2Vn order. If (A) < 0 go on to the next order in sequence.
1,3,5,7	If the overflow indicator is <u>not set</u> to indicate overflow, proceed in the same manner as 20,22,24,26 respectively. If the overflow indicator is <u>set</u>, go on to the next order in sequence.
	Information on setting and resetting of the overflow indicator is given under the heading of each order type concerned. Generally speaking the indicator will be set when overflow occurs in any addition, subtraction, left shift or multiplication, and will be reset so as to cancel any previous indication of overflow in any order which commences by setting (A) =0 or 1/2, and by any order which involves right shifting i.e. right shift, some orders of type 9 and multiplication.
8,+,N,F	Stop
9	Go on to the next order. If a 39 order follows 3S or 3L the effect of the 3S or 3L will be preserved until the next following order.
-	Subtract from the address of the next order,

Values of V	Variant
- (cont.)	during its execution and without altering it in the store, the contents of the right-hand address part of location n.
	A 3- order may be used in front of any order except a magnetic tape order. Note however, that if used in front of a 92 order the address of the latter will be modified in so far as the number of shifts, and therefore the number of characters, is concerned but the character punched will not be altered.
J	Go on to the next order. If a 3J order follows a 3- or 3L order the effect of the 3- or 3L will be cancelled.
L	Same as 3- except that addition is carried out instead of subtraction.

Values of V	Variants
0,4	Copy (A) into location n.
1,5	Clear A. then clear location n.
2	Copy the right-hand address digits of (A) into the corresponding position in location n.
3	Clear A. then clear the right-hand address digits in location n.
6	Copy the left-hand address digits of (A) into the corresponding position in location n.
7	Clear A, then clear the left-hand address digits in location n.
8,+,N,F	Stop.
9,J	Replace (A) and (n) by 1/2.
-	Replace (A) by 1/2 and address digits of right-hand order at n by zero.
L	Replace (A) by 1/2 and address digits of left-hand order at n by zero.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, in all orders that commence by setting (A) = 0 or 1/2.

(2) Orders are read out from the store into the order register for execution in pairs. Hence a type 4 order may overwrite itself in the store without this having any effect on the current execution of the order. Moreover if a left-hand 4V order modifies its right-hand partner this will not have any effect on the current execution of the right-hand order.

Copy (n) into Q

Values of V	Variants
0,2,4,6	Order as above.
1,3,5,7	Clear A, then as above.
8,+,N,F	Stop.
9,−,J,L	Set (A) = 1/2, then as above.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, in all orders that commence by setting (A) = 0 or 1/2.

Divide

6Vn

Divide (AQ) by (n) and place the rounded-off quotient in Q, leaving a residue in A. The rounding-off consists of making q_{39} always equal to one. If $|(n)| < |(A)|$, or if $|(n)| = (A) \geq 0$, the result of the division will be incorrect and the machine will stop with the indicator lamp "÷ HANG-UP" on. Refer to Chapter 2 for details of the residue.

Values of V	Variants
6	Order as above.
3,7	Make (A) = 1/2, then as above.
L,-	Make (A) = 1/2, then as above.
8,+,N,F	Stop.
0	Gives correct result only if <u>divisor</u> and <u>dividend</u> are both positive.
1,4,5,9,J	Give correct result only if <u>divisor</u> is positive.
2	Gives correct result only if <u>dividend</u> is positive.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, in all orders that commence by setting (A) = 0 or 1/2.

(2) A division order can never cause the overflow indicator to be set to indicate overflow, even if division hang-up occurs.

Make $(AQ)' = (Q) \times P(n) + 2^{-39}(A)$, where $P(n)$ is as listed below, and (A) is either the original contents of A, or zero, or one half, according to the value of V.

Values of V	Variants
0	$P(n) = -(n)$
1	$P(n) = -(n)$, (A) = 0
2	$P(n) = -\|(n)\|$
3	$P(n) = -\|(n)\|$, (A) = 0
4	$P(n) = (n)$
5	$P(n) = (n)$, (A) = 0
6	$P(n) = \|(n)\|$
7	$P(n) = \|(n)\|$, (A) = 0
8,+ ;N,F	Machine will stop.
9	$P(n) = -(n)$, (A) = 1/2.
-	$P(n) = -\|(n)\|$, (A) = 1/2.
J	$P(n) = (n)$, (A) = 1/2.
L	$P(n) = \|(n)\|$, (A) = 1/2.

NOTES

(1) The last four variants give a rounded-off product in A.

(2) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, at the start of any multiplication order. At the end of the multiplication the overflow indicator will be set in one case only, namely multiplication of -1 by -1 with (A) initially positive or zero.

V = 0,1,2,3
8,9,+,-

Read a specified number of 4-level characters from the input tape into A, or from A to the output tape or printer according to the setting of the output switch. The address n is interpreted modulo 64, and, so interpreted, must not be zero; if n is zero the machine will stop.

Values of V	Variant
0	Shift all digits of A and of Q n places left, <u>introducing zeros on the right of</u> both A and Q, and discarding digits shifted out from the a_0 and q_0 positions. After each <u>complete</u> group of four shifts replace $a_{36},a_{37},a_{38},a_{39}$ by the binary digits of the next 4-level character on the input tape. Characters having the 5^{th} level punched will be skipped over without being read. If n is an exact multiple of 4, the number of characters read will be n/4. If n is not an exact multiple of 4, the number of characters read will be (n-k)/4 where k = 1, 2, or 3. In this case the order will be terminated by a set of k shifts and this set of shifts will <u>not</u> be followed by any input.
1	Clear A, then same as 0.
2	Output the character determined by the digits

Values of V / 2	Variant
(contd.)	a_0,a_1,a_2,a_3; then carry out a cyclic left shift of the digits of A and Q, exactly as in an 06 order, of four places or such smaller number of places as will bring the total left shift to n places. Repeat until n shifts have been performed and (n+k)/4 characters have been output, where k = 0, 1, 2, or 3 as required to make (n+k)/4 integral.
3	Clear A, then same as 2
8,+	Stop.
9,-	Make (A) = 1/2 then same as 0, 2 respectively.

NOTE

(1) Refer to left shift orders (type 0) for effect of these input/output orders on the overflow indicator.

V = 4,5,6,7
N,J,F,L

These orders will not be operative until the magnetic tape equipment has been installed.

Values of V	Variant
4	Record (Q) on tape, provided that a tape has been started in the record mode by means of an 8L order. The address n (mod. 64) must be 5. In the process of transferring (Q) to the tape equipment for recording, the digits of A and Q undergo a 5-place logical left-shift exactly as in an 04 order. If, at any time during the execution of the five shifts the digit a_0 changes value, the overflow indicator will be set.
5	Clear A, reset the overflow indicator, then same as 4.
6	Fast-rewind a selected magnetic tape. n = 64x where x = 0,1,2 or 3 specifies the tape to be rewound. ~~This order occupies a full word, of which the right-hand half has no significance.~~ A rewind operation will not commence until any preceding search operation on any tape unit has been completed, but once the rewind is started, it goes on automatically in the tape unit concerned, leaving the remaining tape equipment free for use.

Values of V	Variant
7,N,F	Stop.
J	Set (A) = 1/2, reset the overflow indicator, then same as 4. The left shift of 5 places will leave the arithmetic registers in exactly the same state as in an 85 order, but the overflow indicator will be set to indicate overflow.
L	This order, known as Record Block Number, occupies a full word. On the left: $n = 64(x+y)$ where $x = 0,1,2$ or 3 specifies the tape required, and $y = 0$, or 8 specifies forward or reverse, respectively. On the right, the function digits have no significance; the remaining 12 digits are all available to specify the block number. The 8L order is used both to start and to conclude a recording process. The operation it carries out is dependent upon whether or not the specified tape has already been started and upon the sign digit of A. (a) Recording not yet started. If (A) is negative, SILLIAC will stop. If (A) is positive or zero, start the specified tape in the specified direction in the record mode, and proceed to record a block-beginning word labelled with the specified block number. Go on to the left-hand order

8Vn

Values of V

L(Cont.)

Variant

of the next pair.

(b) Recording is in operation

If (A) is negative go straight on to the left-hand order of the next pair.

If (A) is positive or zero proceed to record a block-end word labelled with the specified block number and stop the tape. Go on to the right-hand order of the next pair.

Recording may be disabled, by an appropriate switch setting, for the protection of important tapes against overwriting in error. If the tape selected has been so protected, an 8L order will cause SILLIAC to stop.

V = 0,1,2,3
8,9,+,-

Read a specified number of 5-level characters from the input tape; or output a specified character a specified number of times to the punch or printer depending on the setting of the output switch.

Values of V	Variant
0	Not normally used. A 90 order will give action similar to a 91 order, but omitting the initial clearing of A to zero. The result of this is to give, upon the input of each character, the logical disjunction ("OR") of the digits on the tape and the digits in the corresponding positions of (A) at the instant of input.
1	Normally used with address n = 4. In that case the action is as follows: Clear A; execute a 4-place logical right shift of the 80 digits in A and Q, exactly as in a 15 order; replace the digits in positions $a_{36}, a_{37}, a_{38}, a_{39}$, by the digits from the four normally used levels of the next character on the input tape; replace the digit a_0 by the digit from the fifth level of this character. If n < 4, no input will occur, but the machine will act exactly as in a 15 order. If n > 4, the action described for n = 4 will occur as many times as there are complete groups of four shifts. Such action will be followed by 0,1,2 or 3 right shifts as required

Values of V	Variant
1 (cont.)	to make the total number of shifts equal to n (modulo 64). The address n (modulo 64) must not be zero, or the machine will stop.
2	Output to punch or printer a five-level character once or several times; the particular character and the number of times it is output depend on the address digits as set out below. Also execute an m-place cyclic right shift of the 80 digits of A and Q exactly as as in a 16 order, where m(=n (mod.64)) must not be zero or the machine will stop. The significance of the 10 address digits is as follows (1) The leftmost 4 digits determine the usually used 4 digits of the output character; (2) the rightmost digit determines the fifth digit of the output character; (3) the number m = n (mod64), contained in the rightmost 6 digit positions, determines the number of places by which AQ is shifted, and the number of times, s, that the character is output is given by: $s = \frac{m + k}{4}$ where k = 0, 1, 2 or3 as required to make s an integer. Table 3.4 summarises the "addresses" required to obtain various output characters. "Addresses" are given for the output of one character only. For r characters (r≤16), 4(r-1) should be added to

Values of V	Variant
2 (cont.)	the "addresses" in the table. Any address so obtained[2] may be increased by 2 without affecting the character, or the number of characters, output. The "code equivalent" is the number obtained by treating the character punched on the output tape as a binary number. If a 92 order is preceded by an indexing order (3- or 3L) the resulting modification of the address will change the number of shifts, and hence the number of characters output, but the same character will always be output irrespective of any address modification.
3	Clear A, then same as 92.
9	Same as 91 except begin by setting (A) = 1/2 instead of zero.
-	Set (A) = 1/2, then same as 92.
8,+	Stop.

NOTE

(1) All orders involving right-shifting will reset the overflow indicator so as to cancel the indication of any previous overflow.

(2) Except even addresses where r = 16.

Table 3.4

Code Equivalent	Character Output: Figure Shift	Character Output: Letter Shift	Address of 92 Order: Decimal	Address of 92 Order: Sexa-Decimal
0	0	P	2	002
1	1	Q	66	042
2	2	W	130	082
3	3	E	194	0N2
4	4	R	258	102
5	5	T	322	142
6	6	Y	386	182
7	7	U	450	1N2
8	8	I	514	202
9	9	O	578	242
10	+	K	642	282
11	-	S	706	2N2
12	N	N	770	302
13	J	J	834	342
14	F	F	898	382
15	L	L	962	3N2
16	Delay		1	001
17	#	D	65	041
18	CR/LF		129	081
19	(	B	193	0N1
20	LS		257	101
21	,	V	321	141
22	)	A	385	181
23	/	X	449	1N1
24	Space		513	201
25	=	G	577	241
26	.	M	641	281
27	FS		705	2N1
28	'(prime)	H	769	301
29	:	C	833	341
30	×	Z	897	381
31	Erase		961	3N1

(Note: these "addresses" if increased by 2 will give the same Character.)

V = 4,5,6,7
N,J,F,L

These orders will not be operative until the magnetic tape equipment has been installed.

Values of V	Variant
4	~~Case 1~~. Playback

n = 64 (x+y) + 5

x = 0,1,2, or 3, specifies the tape required.

y = 0 or 8, specifies forward or reverse respectively.

Start the selected tape in the selected direction, in the playback mode, if not already started; copy the first completed word (the first block number word is not read) into Q, at the same time executing a 5-place logical right shift of the digits in A (introducing zeros into the left of A and clearing the overflow indicator); go on to the next order. For playback to continue, execution of the next playback order must have commenced before the completion of the transfer of each word from the tape. When the final block number word is reached this word will be read into Q in the same manner as preceding words; SILLIAC will go on to the left-hand order of the next pair, and the tape will stop. The block number at each end of a block occupies a whole word. The block number appears in the twelve binary digits (4 octal

9Vn
Values of V

Variant

4 (cont.)

digits) on the extreme right of Q. The same four octal digits are repeated on the extreme left of Q in reverse order. Remaining digits are zeros.

If during playback any word from the tape is completed, and there is not at the time any playback order in operation the tape equipment immediately becomes locked so that the SILLIAC will stop if it attempts to execute any playback or record-block-number order, or any search or rewind order specifying the tape unit concerned. The tape will run on to the end of the block and then stop. This locked condition may be cleared either manually or by an "unlock tape" order (see below).

6 ~~Case 2~~. Unlock tape
~~n (modulo 512) ≥ 256.~~

Set the tape equipment so that, as soon as the tape has stopped at the end of the block during which locking occurs, the equipment will be clear for normal use. The unlock-tape order may be placed at any point after the orders which lead to a locked-tape condition, but before the next tape operation is required. If there is no locked-tape condition, an unlock tape order will cause the SILLIAC to stop.

5 Clear A, reset the overflow indicator, then same as 4.

L Search. This order occupies a full word.

9Vn

Values of V	Variant
6 (cont.)	On the left: $n = 64 (x + y)$ where $x = 0,1,2,$ or 3, specifies the tape required, and $y = 0$ or 8, specifies forward or reverse respectively. On the right, the function digits have no significance, the remaining 12 digits contain the block number to be searched for. SILLIAC goes on to the left-hand order of the next pair as soon as the block number has been transferred to the tape equipment. The equipment then proceeds to drive the selected tape in the selected direction until a block is located having the required block number at one end or the other. The tape is stopped at the end of this block, but end-of-search is signalled only if the block has the correct block number at both ends. If end-of-search is not signalled, no further search, or re-wind, can be started on any tape, and no playback or record can be started on the tape concerned, until the equipment is manually reset.
7,N,F,~~4~~ J	Stop. Set (A) = 1/2, reset the overflow indicator, then same as 4.

Increment Add From Q

+Vn

Add or subtract (Q) + 2^{-39} into A, according to the value of V. The address n has no significance.

Values of V	Variant
0	(A) - (Q) - 2^{-39}
1	- (Q) - 2^{-39}
2	(A) - |(Q) + 2^{-39}|
3	- |(Q) + 2^{-39}|
4	(A) + (Q) + 2^{-39}
5	(A)' = (Q) + 2^{-39}
6	(A) + |(Q) + 2^{-39}|
7	|(Q) + 2^{-39}|
9	2^{-1} - (Q) - 2^{-39}
-	2^{-1} - |(Q) + 2^{-39}|
J	2^{-1} + (Q) + 2^{-39}
L	2^{-1} + |(Q) + 2^{-39}|
8,+,N,F	Stop

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, by all orders that commence by setting (A) = 0 or 1/2.

(2) In every case, the overflow indicator will be set if overflow occurs in the result of the addition or subtraction process.

Add or subtract (Q) into A, according to the value of V. The address n has no significance.

Values of V	Variant
0	(A) − (Q)
1	− (Q)
2	(A) − |(Q)|
3	− |(Q)|
4	(A) + (Q)
5	(A)' = (Q)
6	(A) + |(Q)|
7	|(Q)|
9	2^{-1} − (Q)
−	2^{-1} − |(Q)|
J	2^{-1} + (Q)
L	2^{-1} + |(Q)|
8,+,N,F	Stop.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, by all orders that commence by setting (A) = 0 or 1/2.

(2) In every case, the overflow indicator will be set if overflow occurs in the result of the addition or subtraction process.

Copy into storage location n all, or part of, the contents of Q, according to the value of V.

Values of V	Variant
0,4	Copy (Q) into location n.
1,5	Clear A, then same as 0.
2	Copy the right-hand address digits of (Q) into the corresponding position in location n.
3	Clear A, then same as 2.
6	Copy the left-hand address digits of (Q) into the corresponding position in location n .
7	Clear A, then same as 6.
8,+,N ,F	Stop
9,J	Set (A) = 1/ 2, then same as 0.
-	Set (A) = 1/2 , than same as 2.
L	Set (A) = 1/2, then same as 6.

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, in all orders that commence by setting (A) = 0 or 1/2.

(2) Orders are read out from the store into the order register for execution in pairs. Hence a type N order may overwrite itself in the store without this having any effect on the current execution of the order; and if a left-hand NV order modifies its right-hand partner this will not have any effect on the current execution of the right-hand order.

Collate in Q

JVn

Form in Q the digit-by-digit logical product of (n) and the number initially standing in Q.

Va lues of V	Variant
0,2,4,6	As described above.
1,3,5,7	Clear A, then as above.
8,+,N,F	Stop.
9,-,J,L	Make (A) = 1/2, then as above.

NOTE

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, by all orders that commence by setting (A) = 0 or 1/2.

(2) Logical pro ducts:

$$1 \times 1 = 1, \quad 1 \times 0 = 0, \quad 0 \times 0 = 0.$$

Add or subtract (n) + 2^{-39} into A, according to the value of V.

Values of V	Variant
0	(A) - (n) - 2^{-39}
1	- (n) - 2^{-39}
2	(A) - |(n) + 2^{-39}|
3	- |(n) + 2^{-39}|
4	(A) + (n) + 2^{-39}
5	(n) + 2^{-39}
6	(A) + |(n) + 2^{-39}|
7	|(n) + 2^{-39}|
9	2^{-1} - (n) - 2^{-39}
-	2^{-1} - |(n) + 2^{-39}|
J	2^{-1} + (n) + 2^{-39}
L	2^{-1} + |(n) + 2^{-39}|
8, +, N, F	Stop

(A)' =

NOTES

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, by all orders that commence by setting (A) = 0 or 1/2.

(2) In every case, the overflow indicator will be set if overflow occurs in the result of the addition or subtraction process.

Add or subtract (n) into A, according to the value of V.

Values of V	Variant.
0	(A) - (n)
1	- (n)
2	(A) - |(n)|
3	- |(n)|
4	(A) + (n)
5	+ (n)
6	(A)' = (A) + |(n)|
7	+ |(n)|
9	2^{-1} - (n)
-	2^{-1} - |(n)|
J	2^{-1} + (n)
L	2^{-1} + |(n)|
8,+,N,F	Stop.

NOTE

(1) The overflow indicator will be reset, so as to cancel the indication of any previous overflow, by all orders that commence by setting (A) = 0 or 1/2.

(2) In every case, the overflow indicator will be set if overflow occurs in the result of the addition or subtraction process.

LIST OF THE MOST COMMONLY USED ORDERS

ARITHMETIC ORDERS

ADD, SUBTRACT

L5 n	Put (n) in A.
L1 n	Put -(n) in A.
L4 n	Add (n) to A.
L0 n	Subtract (n) from A.
L7 n	Put |(n)| in A.
L2 n	Subtract |(n)| from A.
F5 n	Put (n) + 2^{-39} in A. Used for adding 1 to an integer or an address.
-V n	Same as LV n, but read (Q) for (n). The value of n is irrelevant.

MULTIPLY

75 n	Put (n) × (Q) in (AQ).
71 n	Put -(n) × (Q) in (AQ).
74 n	Put (n) × (Q) + 2^{-39} × (A) in (AQ).
70 n	Put -(n) × (Q) + 2^{-39} × (A) in (AQ).

Use 75, 71, 74, 70 for obtaining double-length products or for multiplying integers.

7J n	Put the rounded product (n) x (Q) in A.
79 n	Put the rounded product - (n) x (Q) in A.

Use 7J, 79 for multiplying fractions when single-length product is required.

To use \|(n)\| in place of (n) add 2 to the V-digit in any of the above arithmetic orders.

DIVIDE

66 n Divide (AQ) by (n). Or
  divide (A) + ε by (n), where $0 \leq \varepsilon < 2^{-39}$.
67 n If (Q) is positive; divide (Q) x 2^{-39} by (n).
6L n Divide 1/2 + ε by (n), where $0 \leq \varepsilon < 2^{-39}$.

Each division order places the rounded quotient in Q.

SHIFT

00 n Multiply (AQ) by 2^n.
02 n Multiply both (A) and (Q) by 2^n.
0- 1 Put -1 in A; multiply (Q) by 2.
10 n Multiply (AQ) by 2^{-n}.
12 n Multiply both (A) and (Q) by 2^{-n}.
1- n Put 2^{-n-1} in A; multiply (Q) by 2^{-n}.
19 n If $n \geq 39$; put 2^{-n-1} in AQ.
  If $n < 39$; put $2^{-n-1}(1 + \varepsilon)$ in AQ, where
  $0 \leq \varepsilon < 2^{-38}$.
06 40 } Interchange (A) and (Q).
16 40 }

The restriction $0 < n < 64$ applies to all shift orders.

MISCELLANEOUS

40 n Store (A) in location n.
41 n Clear (A) and clear location n.
NO n Store (Q) in location n.
50 n Put (n) in Q.
49 n Store 1/2 in location n.

OPERATIONS ON ORDERS, ADDRESS MODIFICATION

42 n — Replace address digits of the R.H. order at n by the corresponding digits of A.

43 n — Clear A; clear address digits of the R.H. order at n.

N2 n — Replace address digits of the R.H. order at n by the corresponding digits of Q.

46 n — Replace address digits of the L.H. order at n by the corresponding digits of A.

47 n — Clear A; clear address digits of the L.H. order at n.

N6 n — Replace address digits of the L.H. order at n by the corresponding digits of Q.

06 20 — Shift the two order pairs in A and Q 20 places to the left, so the R.H. orders move to L.H. positions, the L.H. order in Q moves to the R.H. of A, and the L.H. order in A moves to the R.H. of Q.

L5 n — Put into A the order pair in n.

50 n — Put into Q the order pair in n.

F5 n — Put into A the order pair in n, adding unity to the address of the R.H. order.

+5 n — Put into A the order pair in Q, adding unity to the address of the R.H. order.

3L n — Before obeying the next order, add to its address the number in the R.H. address position at n.

3- n — Before obeying the next order, subtract from its address the number in the R.H. address position at n.

CONTROL TRANSFERS, SENSING, TESTING AND COMPARING OPERATIONS

26 n Transfer control to the L.H. order at n.

22 n Transfer control to the R.H. order at n.

24 n Stop, unless black switch set to IGNORE. If re-started by black switch transfer control to L.H. order at n. If re-started by white switch go on with next order in sequence.

20 n Stop, unless black switch set to IGNORE. If re-started by black switch transfer control to R.H. order at n. If re-started by white switch go on with the next R.H. order in sequence.

When black switch is set to IGNORE, 24 is identical to 26, and 20 identical to 22.

36 n
32 n
34 n
30 n Test the sign digit of A. If zero (i.e. if (A) is positive) do exactly the same as corresponding type 2 order above.
If unity (i.e. if (A) is negative) go on with next order.

37 n
33 n
35 n
31 n Test the overflow indicator. If a zero (i.e. no overflow) do exactly the same as for order 26,22, 24,20 respectively. If unity (i.e. overflow has occurred) go on with next order.

06 nF Move digit a_n into sign digit position a_0 ready for testing.

16(40-n) Move digit q_n into sign digit position a_0 ready for testing.

06 nF Set overflow indicator unless all n+1 digits a_0 to a_n (or q_{40-n} if $n > 39$) are identical.

L3 n Put $-|(n)|$ in A. Result is positive only if (n) = 0.

Control Transfers, Sensing, Testing and Comparing Operations cont'd.

L7 n	Put |(n)| in A. Result is negative and overflow indicator is set only if (n) = -1.
L1 n	Put -(n) in A. Overflow indicator is set only if (n) = -1.
F5 n	Put (n) + 2^{-39} in A. Overflow indicator is set only if (n) = 1 - 2^{-39}.
L0 n	Subtract (n) from (A). Result is positive if (A) $\geq$ (n).
F0 n	Subtract (n) + 2^{-39} from (A). Result is positive if (A) > (n).
F1 n	Put the digit-by-digit complement of (n) in A.
LL n	Put 1/2 + |(n)| in A. Result is negative, and overflow indicator is set, only if |(n)| $\geq$ 1/2.
L- n	Put 1/2 - |(n)| in A. Result is negative only if |(n)| > 1/2.
-V n +V n	Same as LVn and FVn respectively, but read (Q) for (n).

INPUT / OUTPUT

81 4 Read the next sexadecimal or decimal character on the input tape into digits a_{36-39}. Shift (Q) 4 places left.

80 40 Read the next ten sexadecimal characters on the input tape to form a complete word in A. Clear Q.

82 n $1 \leq n \leq 4$; output digits a_{0-3} as a sexadecimal or decimal character. Shift (A) and (Q) n places left in a cyclic shift.

82 40 Output (A) as a set of 10 sexadecimal characters. Interchange (A) and (Q).

91 4 Read the next character on the input tape. The four commonly used levels go to a_{36-39} as in 81 4. The 5th hole, if present, appears in a_0.

92 129+4n Output n+1 carriage return/line feeds.
92 513+4n Output n+1 spaces.
92 1+4n Output n+1 delays.
92 705+4n Output n+1 figure shifts.

$0 \leq n \leq 15$

See page 3-39 for full list of 92 addresses.

MAGNETIC TAPE OPERATIONS

These orders will not become available until the magnetic tape backing store has been installed.

8L 64(x+y) 00 n Record Block Number (full word order)
First time: Start tape unit x in direction y (forward y = 0, reverse y = 8) and record block number n. (A) must be positive.
Subsequent times: If (A) is negative, do nothing. If (A) is positive, record block number n again, stop tape unit x, and jump to next R.H. order.

84 5 Record
Record (Q) on tape started by an 8L order.
Shift (A/Q) 5 places left with a logical shift.

86 64x 00 n Fast Re-wind (full word order)
Re-wind tape unit x. R.H. order is not used.

96 64(x+y) 00 n Search (full word order)
Start tape unit x in direction y and search for a block with the block number n.

94 64(x+y) + 5 Playback
Start tape unit x in direction y, unless already started by same order, and read next full word from the tape into Q, shifting (A) 5 places right with a logical shift. If word read is an end-of-block number jump to next L.H. order.

94 256 Unlock Tape.
Clear tape hang-up condition caused by abandoning playback.

NULL ORDERS, STOP ORDERS

39 n — Null order. Do nothing at all.

3J n — Null order. Do nothing; if preceded by an indexing order (3L or 3S) cancel the effect of this indexing order.

OF n — Stop. Always used to stop the machine at the end of a program.

FF n — Stop. Conventionally used to stop the machine when a programmed test indicates that an error of some type has occurred. (FF = Fault Finish)

_F n — Stop. Conventionally used to stop the machine when a programmed test indicates an error which is almost certainly due to a machine error. An SF stop indicates that the program should be run again.

24 n)
20 n)
34 n)
30 n)
35 n)
31 n) — Temporary Stop. The machine to be re-started with the black switch after a certain action such as placing a new tape in the reader, has been carried out.

ON n — Stop. Conventionally used as a temporary stop (the machine being re-started with the white switch) when the stop control transfer orders are disabled because the black switch is set to IGNORE, or when the action called for concerns the magnetic tape units. The address n may be used to specify which of various possible actions is called for.

LOGICAL OPERATIONS

JO n Form in Q the digit-by-digit product (1×1=1, 1×0=0, 0×0=0) of (Q) and (n). $(Q)' = (Q) \cdot (n)$

F1 n Place in A the digit-by-digit complement (1→0, 0→1) of (n). $(A)' = \overline{(n)}$

+1 n Place in A the digit-by-digit complement of (Q). $(A)' = \overline{(Q)}$

Other logical operations can be performed by using combinations of these orders as detailed in Chapter 7.

ABBREVIATED ORDER LIST

This list is intended to serve as a reminder only. The definitions of many of the orders are incomplete. Reference should be made to the full description in cases of uncertainty. The more commonly used orders are listed, and orders concerned with the backing store are omitted.

$(n)_{r_1-r_2}$ means digits r_1 to r_2 of the contents of storage register n.

Note that $a_r=(A)_r$. 'c' is the order counter.
0 is the overflow register. (0) = 1 when overflow has occurred.

00 n	$(AQ)' = (AQ) \times 2^n$
10 n	$(A^{\sim})' = (AQ) \times 2^{-n}$
19 n	$(AQ)' = 2^{-n-1}$
20 n	$(c)' = n$; next order = $(n)_{20-39}$, stop if b/s to "OBEY".
22 n	$(c)' = n$; next order $\neq (n)_{20-39}$.
24 n	$(c)' = n$; next order = $(n)_{0-19}$; stop if b/s to "OBEY".
26 n	$(c)' = n$; next order = $(n)_{0-19}$.
30, 32, 34, 36 n	If $(A)_0 = 0$, same as corresponding 2V order. Otherwise ignore.
31, 33, 35, 37 n	If $(0)=0$, same as corresponding 2V order. Otherwise ignore.

40 n	$(n)' = (A)$.
41 n	$(n)' = (A)' = 0$.
42 n	$(n)'_{30-39} = (A)_{30-39}$.
46 n	$(n)'_{10-19} = (A)_{10-19}$.
50 n	$(Q)' = (n)$.
66 n	$(AQ) = (n) \times (Q)' + 2^{-39}R$. $R \simeq (A)' + \lceil 2(Q)_0 - 1 \rceil \times (n)$.
74 n	$(AQ)' = (n) \times (Q) + 2^{-39}(A)$.
75 n	$(AQ)' = (n) \times (Q)$.
79 n	$(AQ)' = -(n) \times (Q) + 2^{-40}$.
7J n	$(AQ)' = (n) \times (Q) + 2^{-40}$.
81 4	Input next sexad. to $(A)_{36-39}$.
82 4	Output next sexad. from $(A)_{0-3}$.
91 4	Input next 5--hole character to $(A)_0$, $(A)_{36-39}$.
92 1	Output delay.
92 129	Output CR/LF.
92 513	Output space.
92 642	Output +.
92 705	Output figure shift.
92 706	Output -.
+5	$(A)' = (Q) + 2^{-39}$.
-0	$(A)' = (A) - (Q)$.
-4	$(A)' = (A) + (Q)$.
-5	$(A)' = (Q)$
NO n	$(n)' = (Q)$
N2 n	$(n)'_{30-39} = (Q)_{30-39}$.
N6 n	$(n)'_{10-19} = (Q)_{10-19}$
JO n	$(Q)' = (Q) \cdot (n)$
F5 n	$(A)' = (A) + 2^{-39}$.
LO n	$(A)' = (A) - (n)$

L1 n	(A)' =-(n).
L2 n	(A)' = (A) - |(n)|.
L3 n	(A)' = -|(n)|.
L4 n	(A)' = (A) + (n).
L5 n	(A)' = (n).
L7 n	(A)' = |(n)|.

CHAPTER 4

ROUTINES

One of the first tasks in programming a computation is to break it down into a number of small or medium size operations, each of which forms a fairly distinct logical step in the whole calculation. Such a step may be the evaluation of a function such as a square root, logarithm or cosine. Each step must be carefully defined so that it fits correctly with all the other steps. Having decided exactly what is required of each step, it is possible to proceed with the coding of the individual steps.

This procedure has three important advantages. First, it enables the programmer to concentrate on one part of the job at a time. Secondly, the coding of each step can be tested separately before incorporating it in the program. Thirdly, certain steps are common to a very large number of different calculations; these have been coded and tested once for all, and may be used by any one in the Laboratory.

The coding for one step constitutes a routine. The collection of routines for performing common operations in the SILLIAC is called the library of routines. It includes routines for evaluating many simple functions like those mentioned above, also for printing numbers in various layouts, for integrating differential equations, for solving simultaneous equations, and for many other operations.

A routine, when used to perform part of the work of another routine, is called a subroutine. The simplest type of subroutine is merely a string of order pairs which can be inserted in the appropriate place among the other orders of a program. This is called an open subroutine. However,

it is convenient to be able to transfer control to a subroutine from a number of points in a program, and for this and other reasons another type , called a closed subroutine, is more commonly used.

4.1 CLOSED SUBROUTINES. A closed subroutine is also a string of consecutive words but these do not have to be placed among the other orders of the program. Instead, they may be put in any convenient place in the store. Each time the subroutine is to be executed, control must be transferred to it in a certain special way from any point in the program (the subroutine is said to be entered). It is so arranged that when execution is complete, control is automatically returned to the point from which the subroutine was entered, so that execution of the rest of the program may continue.

In this way a program is seen to consist of several distinct, self-contained blocks, namely the various subroutines and the part of the program (usually called the main program or master routine) which makes use of its subroutines by sending control to them. Internal rearrangement of a routine is usually difficult, but the routines comprising a program can be shuffled about very easily, and this fact makes the coding of large problems much simpler.

It is not necessary to enter each subroutine directly from the master routine; there is nothing to prevent one subroutine being entered from another. A subroutine may itself have subroutines.

In the description which follows, generally two alternatives are given. The reason for this is that during the latter half of 1958, a number of changes and additions

were made to the order code. Library routines[1] which were prepared before this period often call for a "standard entry" in the write-up: where this is the case, the order sequences listed under "old style" are used. The recommended order sequences with the new orders are listed under "new style".

4.2 ENTERING A CLOSED SUBROUTINE. The following two orders must be inserted in the master routine at the point from which a subroutine is required to be entered.

	Old Style	New Style	
k	Any	Any	
	50 k	F5 k	
k+1	26 m	26 m	Subroutine starts at
	Any	Any	$.^{m}$.

Table 4.1

These orders must be, as shown, in the right-hand half of one word and the left-hand half of the next (it may be necessary to waste an order to do this). The $\begin{matrix}F5\\50\end{matrix}$) order contains the address of the memory location in which it is itself contained. The 26 order contains the address of the memory location containing the first word of the subroutine (it actually transfers control to the left-hand half of this word). F5 k will have the effect of making $(A)_{30-39}$ equal to (k+1) x 2^{-39}.

The subroutine, after execution, automatically transfers control to the order following these, i.e., to the right-hand order in memory location (k+1), so that there is no break in the logical continuity of the master routine. There is no need for the user to know how the subroutine does this, but it is not difficult to understand and it helps to

1. Write-ups are available for all routines in the Library.

complete the picture.

4.3 RETURNING CONTROL TO THE MASTER ROUTINE. When the "old style" entry is used, it will be seen that the effect of the 50 order above is to place the order-pair, of which it is the right-hand half, in the Q register. Thus whenever a closed subroutine is entered, the Q register contains a pair of orders of which the right-hand order contains the address of the memory location from which the pair came. Control must be returned to the right-hand order in the memory location following this.

The subroutine uses the information in the Q register to set the address in a transfer of control order (a 22 or 32 order) called the link, which ultimately causes the transfer back to the master routine. It does this by means of the orders shown in Table 4.2[1].

m	+5	0	These orders "plant"
	42	m+n	the link
m+1	--	--	
	--	--	
---	--	--	
	--	--	
m+n	--	--	
	22 (0)		link

Table 4.2

Forming a Closed Subroutine Link (Old Style)

---

[1] Note that, in writing out groups of orders, it is customary to bracket addresses which are altered in the course of the program. This symbolism is not likely to be confused for the use of brackets to signify "contents of", as the meaning is always clear from the context.

The +5 order transfers the order-pair from the Q register to the A register, increasing the right-hand address by 1 as it does so. This address is now the address to which control is eventually to be returned. The 42 order places this address in the link. The link is situated so that it will be encountered by control when the execution of the subroutine is complete. Note that the link should be a right-hand order so that its address may be inserted from the right-hand side of the A register.

It may sometimes be convenient to use a 32 order as a link; this order may then perform a test within the subroutine and return control to the master routine only if some condition is satisfied. Also the above procedure for "planting" the link may be varied if desired, so long as the result is the same. In particular note the variants in sections 4.5 and 4.6 below.

When the "new style" entry is used, the A register will contain a pair of orders of which the right-hand half contains the address of the memory location from which the pair came, increased by 1. The corresponding sequence at the beginning of the subroutine is then merely:

m	42	m+n
m+1	--	--
	--	--
--	--	--
	--	--
m+n	--	--
	22	(0) .

Table 4.3

Forming a Closed Subroutine Link (New Style)

It may sometines be convenient to keep orders in sections of the store unchanged so that, when subjected to a sum check (i.e. when they are all added up in some convenient way) they can be checked against a standard to detect error. When this happens, it is possible to avoid having to plant the link by the use of 3L orders in the following way.

n	40	00	Place link in address 0.
	--	--	
--	--	--	
n+m	3L	00	Add link to following order before it is obeyed.
	22	00	

Table 4.4
Forming a Closed Subroutine Link
(New Style - Alternative)

4.4 PLACING THE ARGUMENT. All subroutines operate on at least one number somewhere in the machine, and there must be agreement between the subroutine and the master routine on the placing of these numbers, and also on the placing of the results of the subroutine operation.

If only one number is operated upon, with the old style of entry the A register is used to hold this number when the sub-routine is entered (since the A register is not being used for anything else at this moment). Similarly, if only one number is produced by the subroutine it is convenient for

the subroutine to leave this number in the A register.

For example, Library Routine R1 is a closed subroutine that finds the square root of the number given in A, and leaves this square root in A. Suppose we wish the SILLIAC to find the largest root of

$$x^4 + 2x^3 + x^2 - c = 0,$$

where c = (10) and satisfies $0 \leq c \leq 1/2$. It can be shown that the root is

$$x = -1/2 + \sqrt{1/4 + \sqrt{c}}$$

Suppose that the constants 1/2 and 1/4 are given in 20 and 21 respectively, and that x is required in 11. Routine R1 will be used to find the square roots; let its first word be in 100. Then the master routine might, under the old style of entry, contain the orders of Table 4.5

```
50   L5   10   Put c in A
     50   50   Enter Code R1
51   26  100   to form √c
     L4   21   Add 1/4 to √c
52   39    0   Waste order
     50   52   Enter Code R1
53   26  100   to form √(1/4 + √c)
     L0   20   Subtract 1/2 to form x
54   40   11   Put x in 11
```

Table 4.5
Master Routine Using a Closed Subroutine
(Old style)

The subroutine itself must use a slightly different method of link planting from that given in Section 4.3, which would destroy the argument in A before using it. It is here necessary to rescue the argument and hold it in a storage location until the link has been planted, as in

Table 4.6

```
 m     40    2    Put argument in 2
       +5    0
m+1    42   m+n   Plant link
       --   --
m+n    --   --
       22   (0)   Link
```

Table 4.6

Modified Link Planting (Old Style)

Under the "new style", the subroutine might have been written in a way which took its argument from Q and placed[1] the function in A. If this were done, the Master Routine might appear as follows. (Note that at the time of writing (12.11.58), no square root subroutine is available in the library with this facility.)

```
50    50    10
      F5    50
51    26   100
      L4    21
52    14    40   Put A in Q.
      F5    52   Enter square root subroutine.
53    26   100
      LO    20
54    40    11
```

Table 4.7

Master Routine using a Closed Subroutine
(New Style)

For other annotations, see Table 4.5)

---

1. Such a practice would lend itself to the use of the mnemonic "Q for question, A for answer".

The link planting section of the subroutine would in this case be somewhat simpler, as the argument would not have to be rescued in the manner shown previously. It might read as shown in Table 4.8, the assumption being that the routine takes its argument from Q.

m	42	m+n	Plant link
	--	--	
--	--	--	
	--	--	
m+n	--	--	
	22	(0)	Link

Table 4.8

Modified Link Planting (New Style)

4.5 PROGRAM PARAMETERS. Sometimes a subroutine is made to carry out slightly different operations on different occasions. For example, one subroutine might be made to print numbers to any number of figures. Another might form the n^{th} root of a number, where n is any positive integer. The value of n, or the number of figures to be printed, is called a parameter of the subroutine.

The value of a parameter must always be specified if the subroutine is to operate correctly. There are two standard ways of specifying parameters; parameters specified in the way now to be described are called program parameters. (The others are called "preset" parameters - see below, Section 4.7.)

A program parameter may be specified in the first half of the word containing the 50 order which is required on entering any closed subroutine. Let us take for example Library Routine R2, which is a closed subroutine for

replacing (A) by its p^{th} root and is entered in the old style. Here p is a program parameter, to be specified as shown in Table 4.9.

k	50	p	Program parameter
	50	k	
k+1	26	m	Subroutine starts at m
	Any		

Table 4.9

Subroutine Entry with Program Parameter (Old Style)

To illustrate this, suppose we wish the machine to compute the positive real root of

$$x^6 - 3x^4 + 3x^2 + 2c = 0 \quad (1/2 \leq c < 1)$$

which is given by the formula

$$x = + \sqrt{1 + \sqrt[3]{1 - 2c}} \ .$$

Suppose that c = (10) and that x is required in 11; suppose also that (20) = -1 and that Routine R2 starts at 100. Then the master routine might contain the program given in Table 4.4.

50	L9	10	(A) = (1/2) - c
	00	1	(A) = 1 - 2c
51	50	3	Program parameter
	50	51	Link
52	26	100	To Routine R2
	L0	20	(A) = $1 + \sqrt[3]{1 - 2c}$
53	50	2	Program parameter
	50	53	Link
54	26	100	To Routine R2
	40	11	x to 11

Table 4.10

Use of Program Parameter (Old Style)

It will be noticed in Table 4.10 that the order containing the parameter (50 3 or 50 2) is actually obeyed by the machine before the subroutine is entered. This is just a waste of time which is not worth avoiding; it does no good and, provided the order containing the parameter is suitably chosen, no harm. It is usual to use a 50 order here; this puts rubbish in the Q register, but the latter is then immediately reset by the following 50 order and no damage has been done.

A more elaborate example of the use of a program parameter is to be found in Library Routine P1 which prints A as an integer or fraction. The entry is by means of the orders of Table 4.11.

k	XY	d
	50	k
k+1	26	m

Table 4.11

Multiple Program Parameters (Old Style)

In Table 4.11 X= 5 or J; if X = 5 and(A) is negative the printed number is preceded by a minus sign, otherwise it is preceded by a space. Also Y = 0 or 2; if Y = 0,(A)x 2^{39} is printed as an integer; if Y = 2,(A) is printed as a fraction (correctly rounded off). The layout of digits is given by d = 10q + s; q digits are printed with a space after the first s.

It will be noted that all combinations of X and Y form harmless orders.

Mutatis mutandis, similar remarks apply under the new style of entry. The descriptions of subroutines will always contain sufficient information to indicate how program parameters are to be presented to the machine.

4.6 INTERPRETIVE ROUTINES. There is a type of subroutine which, instead of executing a single distinctive operation, carries out a whole series of operations. Each operation requires a parameter for its specification, so that the master routine contains a string of parameters, one for each operation. The string may be of indefinite length.

Such subroutines are called interpretive routines. Their use lies in programs that involve actual operations on elements which are not numbers stored in the usual form but may be numbers stored in some special form, or different mathematical entities altogether such as expansions in Boolean algebra. The commonest application is to numbers stored in the so-called "floating point" form (see Chapter 6). There are certain advantages in storing numbers in this form; however, such numbers cannot be added or multiplied in a single SILLIAC operation. A routine is required to handle simple arithmetic, and for this purpose an interpretive routine is used (e.g. Routine A1). Each parameter of the interpretive routine corresponds to one arithmetical operation, just as each SILLIAC order corresponds to one operation in the SILLIAC. Thus, for example, to place in register 6 the sum of the numbers in registers 4 and 5 we write the parameters

85 4
84 5
8- 6

which act in a similar way to the SILLIAC orders

L5 4
L4 5
40 6.

Owing to the close similarity between the parameters and the ordinary SILLIAC orders, the parameters are themselves often referred to as interpretive orders, or merely as orders, "orders", or orders. The "order code"

of the interpretive routine may be described in a similar way to the order code of the machine; it involves reference to an "accumulator" which behaves, for floating point numbers, like the A register of the SILLIAC.

An interpretive routine is entered in the same way as a closed subroutine, but the parameters (i.e., the interpretive "orders") follow the orders causing entry. Finally, a special parameter may be used to cause control to the SILLIAC to be transferred out of the interpretive routine and back to the master routine. Thus, if we imagine that the above example is a complete set of operations to be carried out by the interpretive routine, the master routine would contain the program of Table 4.12.

```
k      Any
       50  k      Enter interpretive routine
k+1    26  m      (old style)
       85  4      Parameters (interpretive orders)
k+2    84  5
       8S  6
k+3    8J  k+4    Send control of SILLIAC to L.H.
                  side of k+4
       00  0
        etc.
```

Table 4.12

Program for Interpretive Routine

A simplified coding scheme using the interpretive technique is described in Chapter 11.

4.7 PRESET PARAMETERS. By making use of a library routine a programmer avoids not only the necessity of writing out the orders, but also the labour of punching them, since a master copy of the tape is kept on file in the Teletype

Room to be copied when required.

If the routine has one or more parameters associated with it, it can meet a wide variety of requirements. However, the use of program parameters consumes both computing time and storage space.

A program parameter has the property that it can be varied from one application to another within the same program. Frequently it happens that, although the ability to choose a value of parameter to suit any particular program is desired, the ability to change the value during the execution of a program is not needed. The value can be set before execution begins; there is no need for the orders which, in the case of a program parameter, set the value afresh each time the routine is entered. A parameter whose value is set before execution begins is called a <u>preset</u> parameter.

The master copy of a library routine must be valid for all values of any preset parameters involved. The fixing of a preset parameter for a particular program must therefore be done after the program is read into the SILLIAC. This operation is carried out by any one of the variants of the Decimal Order Input Routine which are described in Chapter 5.

CHAPTER 5

THE DECIMAL ORDER INPUT

The SILLIAC is a binary machine intended for University research. Since it will be used by a large group of people with various scientific backgrounds, it is important that its use be made as simple as possible. To this end a library of subroutines has been organized and aids in using it have been devised. One of these aids is the group of Decimal Order Input routines such as Library Code X1. These routines have two principal purposes:

(1) To make it possible for programmers to use decimal notation in coding.

(2) To make use of the library easy.

The use of subroutines is discussed more fully in Chapter 4, but one way to characterize a subroutine is to say that it is essentially an extension of the order code of the machine. It is a group of orders used to carry out one or more operations which may be as simple as a square root or as complicated as a complete program for executing the details of floating point arithmetic. In any case the interior structure of a subroutine will depend upon its location in the memory. For example, the left-hand address of the second word in the square root routine (Library Code R1) refers to the eighth word of the routine. If the routine begins at location 10, this address must be 17. But if the routine begins at location 97, this address must be 104. The use of subroutines is very awkward unless problems of address changing can be easily handled.

Another major nuisance in coding is the number

system used by the SILLIAC. In the sexadecimal system using SILLIAC notation (see Chapter 2), the 1024 addresses of the memory are represented by numbers between 0 and 3LL. It is much more convenient if addresses can be expressed in decimal form with SILLIAC doing the necessary converting to the binary system.

With Decimal Order Input routines, every number corresponding to the address part of an order must be written in the decimal system. These decimal numbers are always converted to sexadecimal (i.e. binary) form by any of the Decimal Order Input routines (D.O.I.'s).

Of the several D.O.I.'s available, X1 was the first to be introduced. Routines X2, X12 and X13 offer the same facilities as X1, and some additional facilities as well: they are,however, somewhat longer. The facilities provided by X1 will be described first of all, and the additional facilities afforded by X2, X12 and X13 will be summarised at the end of the chapter. For full descriptions of X2, X12 and X13, the reader is referred to the write-up of these routines. It is desirable for users of SILLIAC to become acquainted with these later routines X2, X12 and X13 as early as possible, and, if sufficient storage space is available, X13 should be used for preference. All of these routines were introduced before the end of 1958, and so could be shortened because of the introduction of the new orders at about that time. However, no attempt has been made to produce more economical versions of these routines at the time of writing.

5.1 RELATIVE AND FIXED ADDRESSES. If we can devise some way of using addresses in decimal form, we never need the 6 characters +, -, N, J, F, L in an address and can use them for other purposes. Let us consider a program

beginning in location 11 of the memory, as shown in Table 5.1.

```
11    40    1
      -5    1
12    L4    13
      46    20
13    51    1
      10    1
```

Table 5.1

Program Beginning at Location 11

The same program, if begun at location 20, would read as in Table 5.2

```
20    40    1
      -5    1
21    L4    22
      46    29
22    51    1
      10    1
```

Table 5.2

Program Beginning at Location 20

The second word has changed. But if we mark each address which depends upon the location of the routine with the symbol L and each address which is independent of the location of the routine with the symbol F, we have the program in Table 5.3

0	40	1F
	-5	1F
1	L4	2L
	46	9L
2	51	1F
	10	1F

Table 5.3

Program With Relative and Fixed Addresses

The addresses ending in L are <u>relative to the location of the first word of the routine.</u>

We can now write each subroutine as if it begins at location zero provided we add to each L-terminated address the location of the first word of the routine when we store the routine. We shall do this by introducing a new symbol +.

5.2 <u>DIRECTIVES.</u> Orders which are written for the SILLIAC use two sexadecimal function digits followed by an address. We follow this convention even in the case of pseudo orders. A <u>directive</u> is such a pseudo order. When we want a group of orders to be placed in memory locations n, n+1, n+2,....., we punch on the tape the directive

$$00\ n+^{1}$$

X1 will recognize this as a directive and will place the orders following the directive in pairs in the locations

---

[1] We shall use small letters to represent decimal quantities and capital letters to represent sexadecimal quantities.

n, n+1, n+2,...., starting with the left-hand side of location n. It will add n to the address of any order terminated with L before placing it in position. The directive 00 n+ is not placed in the memory. Thus, if we have 00 11+ followed by the orders of Table 5.3 we will place in the memory the code given in Table 5.1, while the code of Table 5.2 would be obtained by using the directive 00 20+.

5.3 ASSEMBLING OF ORDERS. Since each address ends with an alphabetic character and since there are always two function digits, X1 can distinguish between address digits and function digits. X1 uses the fixed storage locations 0, 1 and 2 as temporary storage. After two function digits have been read and shifted to a right-hand order position in location 1 the routine converts the decimal address n to binary form n x 2^{-39} and adds it to the function digits. The order pair in location 1 is then stored. Next the contents of 1 are shifted left 20 places and the next order is formed as before. Again the order pair in location 1 is stored in the same address as the previous time, but now the correctly assembled order pair has been stored. The address of the store order which is assembling the program is increased every other time. Thus, the left-hand 20 digits of location 1 always contain the previous order while the next eight digits of location 1 contain the function digits of the order being put in.

When the address ends in + (a directive) it is stored in location 2 and is added to each L terminated address in location 1 before the appropriate order is stored.

5.4 DECIMAL ADDRESSES. The address is converted one digit at a time as it is read from the tape. This kind of

conversion makes it unnecessary to write non-significant zeros, so orders may have the form L5 7L, 40 1021F, 26 L, etc. Indeed, we are not restricted to addresses smaller than 1024 and may use anything we please. If the address exceeds 4095 it will add to the function digits (remember that there are two unused digits between the function digits and the address), but even this may be made use of as we shall see below in Section 5.6.

5.5 STARTING THE PROGRAM. After X1 has placed a program in the memory we must start the program. This is done by using the terminating symbol N. The symbol N causes the order which it follows to be obeyed. It is used with a control transfer order which must appear on the tape as if it were a left-hand order. It will never be stored in the memory, other than as the right-hand order in location 1.

For example, the order 26 pN will be obeyed and will transfer control to the left-hand order at address p. Any unconditional transfer order may be used. The order 20 qN will stop the computer, and when the computer is started again control will go to the right-hand order at address q. The addresses p and q are fixed addresses.

Control can also be transferred using a relative address, for the previous order with its adjusted address is always in the left-hand side of location 1. Therefore the two orders 22 rL 26 1N will cause control to be transferred to address r relative to the last directive if 22 rL is the right-hand order of the last order pair, i.e., if the phase is correct.

5.6 INPUT OF DECIMAL FRACTIONS. In section 5.4 it was pointed out that an address is formed and added to the

function digits. This address could be as large as $2^{39}-1$ without getting into the sign digit, and any positive integer n could be input as $n \times 2^{-39}$ by letting the left-hand order and the right-hand function digits be zero. For example, the "order pair" 00F 002896F would appear as 2896×2^{-39}. Hence a <u>12 digit right-hand address smaller than 2^{39}</u> (about 5.5×10^{11}) could then be converted to a decimal fraction by multiplying it by $2^{39}/10^{12}$.

This is done when the terminating symbol J is used. For example, the characters 00F 002960 0000 0000J would cause the quantity 0.296 to be placed in the memory. Zeros at the end cannot be omitted, but preceding zeros can be omitted. <u>Do not omit function digits.</u> Remember that this is essentially an integer input. The range can be extended by using the left-hand function digits, for they are simply a number to which the right-hand address is added after being multiplied by $2^{39}/10^{12}$. The left-hand function digits 40, 80, NO represent 1/2, -1, -1/2. Thus numbers could be input as follows:

-0.8888	8888	8888	as	80F	00	1111	1111	1112J
0.7854	3216	0000	as	40F	00	2854	3216	0000J.

This is not an efficient way to read decimal fractions into the machine, but it is convenient for occasional numbers scattered through the program. The normal way to input numbers is to use one of the input subroutines designed for the purpose.

5.7 <u>PRE-SET PARAMETERS. MODIFICATION OF ORDERS.</u>

The remaining unused alphabetic sexadecimal character is -. This is used to modify orders by using pre-set parameters, called S-parameters (as S is the letter-shift

equivalent of -). The symbol - differs from the other terminating symbols in that it is <u>always followed by another single character</u> which may be any of the 13 sexadecimal characters 3 to L. The termination -D on an address causes the content of location D = 3, 4,....., F, L, to be added to the order while it is in a right-hand position as described in Section 5.3 and before it is placed in the memory. This is for either right- or left-hand orders of completely assembled order pairs. For example, if location J contains 7×2^{-39}, the order pair L5 20-J 40 30-J will be modified to read L5 27F 40 37F before being stored. This facility is very convenient for using parameters with a program because the program can be written with orders of the form L5 -3, L4 -4, and if locations 3 and 4 have been previously set the parameters will be added appropriately as the program is read into the machine.

Many examples of pre-set parameters can be found in the library. In Routine P6, Single Column Print, the parameter 00F 00mF is used to specify the printing of m decimal digits and must be placed in location 3 before Routine P6 is read. Thus, if we wish to place Routine P6 in locations beginning with 50 and to print 7 decimal places, the pertinent part of the program tape would read

00	3+	Directive
00	F	Place 7×2^{-39} in
00	7F	location 3
00	50+	Directive
Code P6		Place Code P6 beginning with location 50.

5.8 <u>EXAMPLE OF USE OF DECIMAL ORDER INPUT.</u> X1 has 25 words and occupies locations 999 to 1023 in the memory.

It is placed in these locations with its own bootstrap input which then transfers control to X1 so that it can take over the control of program input. Let us consider the following simple example:

> Compute to 10 decimal places the square roots of $\pi/10$ and e/10 using the Square Root Routine (R1) and the Single Column Print routine (P6). We have the following data:
>
> (a) Routine R1 - 9 words, closed, finds square root of argument placed in A and places answer in A.
>
> (b) Routine P6 - 14 words, closed, prints words in A to m decimal places followed by carriage return and line-feed. Contents of address 3 must contain $m \times 2^{-39}$ as Routine P6 is input.

In this program, the code for which is given in Table 5.4 on the following page, we have scattered the orders through the memory to indicate how directives are used. Notice that the arrangement on the tape is arbitrary after X1 except that the parameter in address 3 must be in place when P6 is being input. The parameter is used in word 6 of P6, this word being

```
19  63-3
50     F  .
```

Thus the address of the 19 order was set to 9 (73 = 9 mod 64) for counting the number of digits to be printed.

Notice also that the directive 00 560+ can be changed so that the words following it are placed differently but that no other change need be made to move these words

MEMORY LOCATION	PROGRAM TAPE	REMARKS
999 - 1023	Decimal Order Input	Routine X1
	00 10+	Directive
10 - 18	Square Root Routine	Routine R1
	00 3+	Directive
3	00 F	Parameter
	00 10F	
	00 30+	Directive
30 - 43	Print Routine	Routine P6
	00 50+	Directive
50	00 F	π/10
	00 314159265359J	
51	00 F	e/10
	00 271828182846J	
	00 560+	Directive
560	L5 50F	π/10 to A
	50 L	Link
561	26 10F	To Routine R1
	50 1L	Link
562	26 30F	To Routine P6
	L5 51F	e/10 to A
563	39 F	Waste Order
	50 3L	Link
564	26 10F	To Routine R1
	50 4L	Link
565	26 30F	To Routine P6
	0F F	Stop
566	39 F	Waste
	26 L	Start Program at relative location 0 after stop.
	24 1N	

Table 5.4

Use of Decimal Order Input

because the program was started using a relative address. The waste order at location 566 is required so that 24 1N will have a left-hand location.

5.9 USE WITH INTERLUDES. RETAINED DIRECTIVE. An interlude is a computation performed during the input of a program, the input being interrupted and then resumed.

A tape bearing a library routine may begin with an interlude which is placed in locations destined eventually to hold the routine itself. When the words of the interlude have been read control is directed to it, the interlude is executed, and then input is resumed. The next part of the tape carries the routine itself which is written over the interlude. The purpose of the interlude is usually to prepare some orders or constants required for the routine. For example, a printing routine, where the number of digits printed is determined by a preset parameter, may use an interlude to compute the roundoff constant.

Input is resumed after the interlude by transferring control to the left side of location 999 (3F7 Sexadecimal). Either the first word on the tape must contain the needed directive 00 m+ or Q must contain m x 2^{-39}.

If, upon resuming input, it is desired to retain the last used directive, control should be transferred to the right-hand side of 1014 with m x 2^{-39} in A. The next words on the tape will be placed in m, m+1, ..., retaining the previous relative address. The interlude must not use location 2.

5.10 STOPPING THE TAPE. The order 20 1019N on the tape will stop the computer and will have no other effect. Upon being started the tape will continue being read from

where it stopped.

5.11 PLACING THE DECIMAL ORDER INPUT. BOOTSTRAPS.

Up to this point we have not said how the Decimal Order Input is itself put into the SILLIAC. It occupies locations 999 to 1023, the last 25 positions of the electrostatic memory, and it is placed in the memory with a bootstrap input routine.

With panel switches we can clear the control counter to zero and place the order pair 80 40F 40 F (80 028 40 000 in sexadecimal) in the order register. The code in Table 5.5, which must be written with sexadecimal addresses, will then place the Decimal Order Input in the memory.

80 028
40 001

80 028
40 002

19 026
26 000

80 028
40(000)[1]

L4 001
40 001

80 028
40(3F6)

Table 5.5

Tape for Bootstrap Input Routine

---

[1] Parentheses are often placed around addresses which change during the course of a program.

This bootstrap actually places the 3-word routine (shown in both sexadecimal and decimal forms) of Table 5.6 in the memory and starts it

0	L4	001	0	L4	1F
	40	001		40	1F
1	80	028	1	80	40F
	40	(3F7)		40	(999)F
2	19	026	2	19	38F
	26	000		26	F

Table 5.6

Memory Contents for Bootstrap Input Routine

at location 1. Clearly it will take the next words on the tape and start putting them at location 999F. To stop the input we place the order pair 22 3L- 00 001 (that is 22 1019F 00 1F) on the tape so that it gets put into location 0. The control will be transferred to the right-hand side of location 1019F and X1 will be started.

This bootstrap input may be used with any code and it, or something like it, must be used to input programs when the Decimal Order Input routine being used has been overwritten.

The term bootstrap start is often used for tapes which are started by setting the order register to 80 40F 40 F and control counter to zero.

5.12 S PARAMETERS AS DIRECTIVES. A program commonly consists of a number of routines and subroutines some of which have been taken from the program library while others

have been written and tested by the programmer before inclusion in the program. Some of these routines play the role of sub-routines which are called in by one or more of the other routines. Thus, for example, the library routine R1 (square root) may be used by several of the routines comprising a full program. When such a program is being written it is not generally known in advance how much store space each of the routines will use, and therefore it is not possible to allot store locations for each of the subroutines. Consequently a relative address must be used in all orders in one routine which refer to another routine. This can be achieved by making use of the - termination with X1 and X2, and with routines X12 and X13, where special facilities are made available to cater for it. The technique to be used with X1 and X2 will be described here.

Each routine (or subroutine) should be allotted a number from the range 4 to 15. This number specifies the store location at which the address of the first word of the routine may be found. This allows for 12 routines, which is generally ample. We cannot use the numbers 0, 1, 2 as these store locations are used by the D.O.I.'s 3 can be used but it is preferable to leave it free for the interludes described in section 5.13. Routine number 7, (say) should then commence with the following interlude which plays the role of a directive,

```
00 2+
50 7F
26 999F
26 2N.
```

This one word interlude may be stored in any free store

location; we have suggested location 2 because it is always free. Location 2 is used by the D.O.I. to hold the directive which is only used when an order terminated by L is read from the tape. When, as in this case, no L terminations are used (the interlude consists of only two orders both terminated by F) the directive in location 2 can be overwritten without harm.

This interlude transfers control to location 999 with the number from location 7 in the Q register. This results in this number being used as a directive for the following orders on the tape.

A complete program tape made up in this way has, immediately following the D.O.I., the addresses of all the routines in the form:

$$\begin{array}{ll} & \text{OO } 4+ \\ \text{OOF} & \text{OOOOOOO}n_4\text{F} \\ \text{OOF} & \text{OOOOOOO}n_5\text{F} \\ \text{OOF} & \text{OOOOOOO}n_6\text{F} \qquad \text{etc.} \end{array}$$

where n_4 is the store location of routine number 4, and so on. The superfluous zeros in front of each address enable alterations to the addresses to be made with the use of a hand punch and thereby saves the need of making up a new tape. As many zeros as may be considered necessary can be provided in this way. Note that the changing of the values of these addresses is the only alteration required on the tape in order to move the routines about in the store.

If each routine is preceded by an interlude of this form its location in the store is completely determined by the address preset in the appropriate store location. Furthermore if, say, routine 8 has to call upon routine 5,

which is a closed subroutine, the appropriate orders in 8 will be

```
p     Any order
      50 pL
p+1   26 -5
      Any order.
```

The order terminated by -5 will have the number in location 5, i.e., the address of the first word of routine number 5, added to it by the D.O.I. Note that the number of the routine must here appear as a sexadecimal number, hence the restriction to numbers less than 16.

5.13 MODIFICATION OF ROUTINES. It is often necessary to place a modification tape at the end of the program tape to modify one or more orders in the program. If the new order is simply located by a directive (00 n+) it cannot use a relative address and the value of the directive must be altered if the routine which is modified is moved in the store. If the routine which is to be modified has been assigned an S-parameter as set out in Section 5.12 the modification can be performed in a way which is independent of the location of the routine. For example, suppose the 25th word of routine number 9 must be changed to read 26 -6 40 8L. This can be done by means of the interlude

```
      00 2+
40 25-9 26 999F
26 -6   40 8-9
      L5 3N.
```

The new order is first stored in location 3 and is then moved to its proper position by the interlude. Note that the terminating symbol L must be replaced by the appropriate

S termination.

This interlude violates the normal rule that an order terminated in N must be a transfer of control order. Orders terminated by N are stored by the D.O.I. in the right-hand side of location 1 and then obeyed. If such an order is not a transfer of control order the next order will be taken from the left-hand side of location 2 which is precisely what we want to happen in this case.

This interlude can be placed anywhere on the tape as long as it comes after the routine which it modifies. It will normally be placed on the end of the program tape if the modification is of a very temporary nature, or immediately after the routine it modifies (by a suitable editing process) if the modification is to be retained for any length of time.

If several successive orders have to be modified the following interlude should be used.

```
        00 2+
40      2F L5  3F
22 1014F 00 n-m
        L5 mN
```

The new orders.

This interlude acts as a directive for the orders which follow it on the tape. The above example takes the contents of location m and stores it in location 2. Thus the address of the first word of routine number m becomes the directive. This overwrites the first word of this interlude which is also stored in location 2, but, as this word has already been taken into the order register at this stage it will still be obeyed. The contents of location 3 are now taken into the A register and control is transferred to the D.O.I. at the right-hand side of location 1014. This

means that the address in the right hand side of location 3, i.e. the address n-m which is the location of the n^{th} word of the routine, specifies the location at which the first word read from the tape will be stored. Thus the orders following on the tape will overwrite the orders of routine number m starting at word n. These orders can be written in exactly the same form that they would have if they were being input as part of the routine which they are overwriting, i.e. **L** terminations have their usual significance.

Note that the number m must appear as a decimal number in the order L5 mN, but as a sexadecimal number where it follows the - in the "order" 00 n-m.

These techniques are required with X1, X2 and X12. X13 provides the means for making modifications more simply (see below).

5.14 STORING ROUTINES IN SEQUENTIAL LOCATIONS. It is sometimes convenient to be able to arrange that the routines on the program tape should be stored in sequential locations so that a directive is required only for the first routine. This cannot be done simply by punching the routines one after the other with nothing between, for then the address added to all addresses terminated in L would be the address of the first word of the first routine instead of the first word of the routine being input. What is required is a short interlude placed between the routines, which serves the purpose of bringing the directive "up to date". A useful technique is provided by the following interlude which would follow a routine containing p words and be immediately followed by the next routine.

```
p-1   Last word of routine
p     L5 p+1L
      42 2F
p+1   22 1014F
      26 pL
      26 1N.
```

The interlude is stored in the locations that will be occupied by the first two words of the next routine. Control is transferred to the first word of the routine which plants the number pL in location 2, thus making it the new directive, and then transfers control to the D.O.I. at the right-hand side of 1014 so that the number pL also specifies the location at which the first order pair read from the tape will be stored.

Special arrangements are available with X13 to facilitate sequential storage of routines (see below).

5.15 S-DIRECTIVES. A normal directive is of the form

OO n+

where, in practice, n is never zero. X12 incorporates orders which test to see whether n is zero. This enables us to use a new type of directive of the form

OO +D

where D is a single sexadecimal digit. When X12 finds that n is zero it reads the additional digit D and uses it to select the pre-set parameter in location D as the directive. Thus this directive is equivalent to

OO d+

where d is the decimal integer pre-set in location D.

These directives are intended to be used in the following way. Each routine comprising the full program is assigned a sexadecimal number D in the range 3 to L. (However, it is better not to have a routine numbered 3 because the S3 parameter is used for a special purpose by several library routines. Also it is advantageous to preserve the routine location S-parameters throughout the running of the program and location 3 is used as temporary storage by many library routines.)

Orders in one routine can then refer to another routine by use of the -D termination as long as the addresses of

the routines are correctly pre-set. Using these pre-set addresses to determine directly where each routine is to be stored makes it very easy to vary the positions in the store in which each routine is to be stored. The main body of the program tape contains no explicit statement as to where the routines are to be stored, and need therefore not be altered.

If, by reason of a programming error, the parameter d has not been pre-set in location D, i.e. if (D)=0, then X12 will read a further sexadecimal character and treat it as D, and so on until a non-zero (D) is obtained. This can obviously lead to absurd results, but it does enable us to punch an S-directive as

00 +000...00D

where any number of zeros may appear between + and D. This enables the value of D to be altered easily.

The directive 00 +1 should never be used.

The directive 00+2 has the effect of taking the current directive (stored in 2) as the new one, i.e. it results in the new routine overwriting the previous one. This can be useful when a routine starts with an interlude.

A similar technique can be used to facilitate starting a program. A normal start order is of the form

2B nN

where B = 0, 2, 4 or 6. In practice n can never be zero. By using a zero value of n which is detected by X12, we can use a new type of start order of the form

2B ND.

This order is interpreted as

where d is the integer pre-set in location D, with two exceptions:

(1) If d = 0 but D ≠ 0, i.e., if the pre-setting of the parameter has been neglected, a further digit is read as D just as in the case of the directive.

(2) If D = 0, d is taken as being 984. This is to facilitate the comparing of the check sum - see below.

The order 2B N1 should never be used. The order 2B N2 causes control to be transferred to the first word of the routine just read.

Note that the order 26 1014F,which causes the D.O.I.'s X1 and X2 to take as the new directive the right-hand address in A, cannot be used with X12. The same function with X12 is performed by the order 22 999F.

5.16 SUM CHECK FACILITY. X2 and X12 provide a means of ensuring that a program has been read in correctly by SILLIAC. X12, which has facilities not available with X2, should be used when this facility is required, unless space is very much at a premium, in which case the fact that X2 is four words shorter will count to its advantage. For this reason, pertinent details of X12 only are given here: the means of obtaining equivalent facilities with X2 are described in the write-up of that routine.

X12 forms the sum of all orders and pseudo-orders read from the program tape and can be instructed to compare the sum so formed with a sum provided on the end of the tape. It thus provides a check on the operation of the

tape reader and guarantees that the full program has been correctly read into SILLIAC before the program is actually started.

The sum formed by X12 is stored in location 983 (the first word of X12) and can be operated upon in the following ways.

The order

26 NO

placed on the program tape instructs X12 to read a further five sexadecimal digits from the tape and to compare them with the sum in 983. After the comparison is made location 983 is cleared ready to receive the sum of any further orders. The order

22 988N

on the program tape instructs X12 to clear location 983 without making any comparisons. Both these orders can be placed anywhere on the tape where they will appear as left-hand orders. They do not affect the current directive.

If the comparison of the check sum, under the control of the 26 NO order, indicates a read-in error, SILLIAC stops on a -F order (-F = SF = Sum Failure) in the left-hand side of location 988 (3JN in sexadecimal) after printing out five sexadecimal characters. If then restarted with the white switch it will continue reading in as if the sum check had not failed.

The normal method of using the sum check feature of X12 is to place at the end of the tape the following (the erase characters are not essential; their use is explained later):

26 NO
Two or three erases
00000
Two or three erases
Run of delays
24 999N[1] or 20 1019N
Run of delays
The start order (normally 26 ND).

SILLIAC will stop reading immediately after reading the five zeros and will print out five sexadecimal characters, and then stop on the -F order. On restarting with the white switch SILLIAC will proceed to the 24 999N order and the normal procedures of code checking can be followed.

When the program is ready for production running, the check sum should be punched onto the tape. The five zeros which were left on the tape for this purpose must be located (this is greatly facilitated by the erase characters punched at either side) and the five sexadecimal characters printed or punched out by SILLIAC on the last successful code checking run should be punched over them. When read into SILLIAC again the tape will now go straight past the sum check order without stopping (unless a read-in error has occurred).

Alterations can still be made to the program without providing a new check sum if they are made by means of a modification tape which is read after SILLIAC stops on the 24 999N order. Also the 24 999N order can be changed to a 26 999N order (to eliminate the stop) without affecting the check sum.

If any alteration to the program is made on the program tape, thus requiring a new check sum, the tape should not be read in with the old check sum still on it

1) See page 5-36. 24 999N destroys the directive.

but with five zeros again, for the number printed out by SILLIAC is the difference between the true sum of the orders and the sum on the tape.

When a lengthy program is prepared it is sometimes convenient to prepare and test the various routines separately. When this is done, it is advisable to attach a check sum to each routine. To facilitate this technique X12 has been designed so that the check sum of a routine is independent of the position in the store in which it is placed, as long as the S-directive 00 +D is used. If a normal directive is used, it will be added to the sum and changing it will change the sum. A directive can be omitted from the sum by placing 22 988N after it.

The check sum is automatically cleared to zero when X12 is first read in and also by each of the sum checking operations. Hence the clear sum order 22 988N need only be used where orders, or pseudo-orders, which should not be included in the check sum actually appear on the tape.

Orders are added to the check sum in the form in which they appear on the tape, i.e., before the addition of the directive (for L termination) or an S-parameter (for -D termination). However, the terminating symbol itself is added to the sum so as to check that it is read correctly. The check sum is therefore unaffected by the location of the routine in the store and by the values of pre-set parameters.

5.17 FLOATING ADDRESSES AND TAGGED ORDERS.* X13 provides, in addition to the normal D.O.I. facilities, a number of facilities which are extensions of those already described. For example, it allows a virtually unlimited number of S-parameters to be used. Also, it has a

facility which makes it unnecessary to pre-set directives either as absolute addresses (as with X1) or as S-parameters (as with X12): the S-parameter can be planted automatically. Reference should be made to the X13 write-up for details.

X13 has two other facilities which deserve special mention - and these are described as Floating Addresses and Tagged Orders. To understand their usefulness, we must look closely at the reason why we must keep a tally of the location of each order written down as we code.

When coding a program which is to be read in by the conventional D.O.I. it is necessary to keep a continuous tally of the location of each order relative to the first order of its routine, and also to allocate orders to right- and left-hand sides and to insert "waste" orders wherever the natural sequence of left-right must be broken. Now, there are normally only three reasons why this tally must be kept, and they are:

(1) We must be able to refer to one order by another one, and therefore must know the (relative) address of all orders to which such reference must be made. The reference may be in the nature of a transfer of control or an arithmetic operation upon the order.

(2) We must know whether an order is to go into the right- or left-hand side.

(3) We need to know how many store locations are occupied by each routine so that we can
   a) allot space in the store to each routine,

b) determine whether the program will fit into the store, and

c) locate any given order in the store during code checking runs.

With the use of X13 we eliminate the need for counting the orders, although we must still pay attention to points 3b) and 3c).

The "tag" of an order (or pseudo order[1]) is a number assigned to it, this number being selected at will and bearing no relation to the actual location of the order. Then any other order which must refer to this one can do so either by referring to its true address (relative or fixed) or by referring to its tag. Thus, when an order has been tagged, the store location of that order can be referred to by the tag; hence we speak of the true address of the order as a "tagged address".

If an order refers, via a tag number, to another order which is placed after it on the program tape, then its true address digits cannot be determined until the tagged order has been read by X13. When an order is waiting in this way for its address digits to be completed, we say that it has a "floating address". When a tagged order is read by X13 it is able to complete the floating addresses of all preceding orders which refer to this tag. It also keeps a record of the tagged address in case further floating address orders referring to this tag are read.

When we tag an order, we specify three things:

---

[1] A "pseudo-order" is a constant input as an order by means of one of the D.O.I.'s.

(1) The tag number itself. This is an arbitrarily assigned positive integer which must be less than 2^{11} (= 2048) and should not be zero except in a special case discussed later.

(2) Whether the order will be referred to by order(s) in other routine(s) or not. If the tagged address is not required by orders in other routines, then X13 can "forget" the address as soon as the current routine has been read in.

(3) Whether the order must be stored as a left- or right-hand order. If the tagged order does not naturally fall on the side specified, X13 will insert a pure "waste order" immediately before it. The waste order is a

39 order.

The tag, which always precedes the order, is indicated by the symbol ≠ . This is followed by the tag number itself, which is terminated by the character L or -. L indicates that the tag is only wanted within the current routine, and - indicates that the tagged order may be referred to by orders in other routines, the tag being referred to as an L-tag or an S-tag. The L or - is followed by the digit 2 or 6. A 2 indicates that the order must be stored as a right-hand order, and a 6 that it must be a left-hand order. (To assist in remembering these numbers, note their connection with the meaning of the 2 and 6 in such orders as 22, 26, 42, and 46.)

Examples of tagged orders are:

```
≠1L6  L5  F
≠2L2  40  ()F
≠23-2 -5  10F
```

A constant in the form of an order pair may be tagged, e.g.,

```
≠4-6 00F 00 100F
≠17L6 40F 00 3322 0000 0000J.
```

Note that in writing orders we make no attempt to group them as order pairs, but that it is still convenient to write constants as order pairs. Note also that a tag on a constant must always end in 6, thereby specifying a left-hand location for the first half of the constant - the second half will then automatically be placed alongside it.

Note that, unlike the standard D.O.I., X1, X13 cannot handle integers which are so large that they overflow into the left-hand order. This means that an integer n can be punched in the form

00F 00nF

only if n is less than 2^{20} (=1048576). A larger integer can be expressed in the form

$$n = a.2^{20} + b$$

and punched as

00 aF 00 bF .

A floating address is indicated in a similar way to a tag. The symbol ≠ is followed by the tag number and either L or -. There is no need here for an indication of the side of the tagged order, so we do not follow the L or - by a 2 or 6. However, the - must be followed by one or three

sexadecimal digits indicating the routine number of the routine which contains the corresponding S-tagged order[1]. Examples of orders with floating addresses are:

```
                    26   #=1L
                    42   #=2L
          #=14L6    L5   #=23-7
    or    #=14L6    L5   #=23-007 .
```

(The orders given here as examples refer to the tagged addresses given earlier.) The 7 (or 007) after the - in the third example indicates that the corresponding S-tagged order is in routine number 7. A tag has been placed on the third example to indicate that the one order may contain a floating address and be tagged as well.

Note that the symbol #= is the only fifth-hole character read by X13, all other fifth-hole characters are completely ignored.

Although the tag numbers may be chosen at will, it is generally advantageous to allot them serially, starting from 1. The L-tags and S-tags are independent of each other and so we may start at 1 for both of them and continue each sequence serially. The S-tags of one routine are also independent of those of other routines. Thus, if we have n routines, we may have 2n different independent sequences of tag numbers.

The commonest type of coding error that may occur in the tagging of orders is the neglect to tag an order which should be tagged, thereby leaving a floating address without a corresponding tagged address. The serial allocation of

---

[1] The ability to use an unlimited number of S-parameters requires that in cases where addresses in excess of 15 are referred to, a three-digit group is employed. See full write-up of X13.

tag numbers helps to eliminate this error, but even if it still occurs, X13 has a built-in check which will pick up the error; more of this later.

A zero tag number should never be used in an S-tag. The zero L-tag has a special use and, unless the user is very cautious, should not be used in any other way. The zero L-tag has been specially designed for use with link setting orders. A link setting order does not need to be tagged in the normal sense unless it is referred to by another order, but we do need to be able to specify that it should be a right-hand order and also that its address digits should be equal to its actual address. This is done by punching the order in the form:-

```
#L2  50 #L.
```

If the link setting order is to be referred to by another order, we give it a normal tag instead of a zero tag, thus

```
#nL2  50  #nL.
```

A full word link setting order can be specified in a similar way. For example, the library print routine P1 requires a link order pair of the form

```
p  XY dF
   50 pL .
```

If using X13 we would punch this as

```
#L6 XY dF
     50 #L
```

or as

```
#nL6 XY dF
     50 #nL
```

if the link order is to be referred to by another order.

The advantage of using the zero tag for such orders lies in the fact that a zero L-tag occupies no space in the directory. The zero tag is stored in the directory, but is over-written by the next directory entry. It is therefore "remembered" by X13 only until the next directory entry is made. A zero tag should never come <u>after</u> the floating address which refers to it.

A single order may be allotted any number of tags. The need for this may occur in one of the following ways:

In the process of correcting or modifying a program tape it may be found that two (or more) orders, both of which refer to the same tagged order, have different tag numbers in their floating addresses. If it is found easier to allot <u>both</u> the tag numbers to the order referred to, than to change the tag number in one of the floating addresses, this may be done. For example, we may have

```
#13L2  #4-2  L5  1F
```

when the order L5 1F may be referred to by either the #13L tag, or the #4- tag. One may allot any number of tags in this way, either L- or S-tags, but all the tags must specify the same side, i.e., either all left-hand or all right-hand sides.

There are cases where it is essential to place at least one waste order between two successive functional orders. A common example is:

```
          42  #nL
#nL2 26  ()F   .
```

If the 42 order is a left-hand order, the "switch" order will be its right-hand partner and the planting of the address will have no effect the first time the switch order is obeyed. To prevent this happening we can specify

```
          42  #=nL
    #=L6
    #=nL2 26  ()F  .
```

If the 42 order is a left-hand order, X13 will follow it with a waste right-hand order when it reads the #=L6 (so that the following non-existent order can be placed on the left as specified by the zero tag). Another left-hand waste will be stored when the #=nL2 tag is read, yielding

```
p     42 p+1F
      39 p+1F

p+1   39  p+1F
      26 ()F .
```

If the 42 order is a right-hand order the zero tag will have no effect and we will have stored

```
p     -- --
      42 p+1 F

p+1   39  p+1F
      26 ()F  .
```

Thus in both cases the required result is obtained with the minimum number of waste orders.

If the programmer should so desire a sequence of tags alternating left-hand, right-hand may be placed before an order so as to cause a sequence of waste orders to be stored.

The use of X13 greatly simplifies the work of coding a program in that it enables the complete code to be written without any attention being paid to the counting of orders. A program which makes full use of the facilities provided by X13 has no normal addresses with L terminations at all -

they are all replaced by tags. The correction or modification of a program is also made very easy as the insertion of additional orders or the deletion of orders from a program is very easily done. As no actual addresses are specified anywhere in the program tape the insertion or deletion of an order requires no other changes to be made.

Nevertheless, after a program has been prepared it is strongly advised that the orders in each routine should be counted (allowing for the necessary waste orders) and the position of each routine in the store be ascertained. It is good practice to take a print-out of all the S-parameters at the end of the first code check run, and again after each run with a corrected or modified program. This enables the location and the length of each routine to be ascertained without any chance of error such as may easily occur in a manual count of the orders. The S-parameters can be easily printed out with the post-mortem routine C4 or C5 (see Chapter 8). To enable this to be done it is obviously essential that the S-parameters should not be overwritten during the operation of the program. If store space is at a premium it may be necessary to use the S-parameter locations as working space. This has no harmful effect except in that they are then no longer accessible to a post-mortem routine.

One of the commonest forms of error in coding a program for use with X13 is in the inadvertent omission of a tag. This leaves a floating address order without a corresponding tagged order. If this occurs with an L-type tag it is automatically detected by X13 after reading the next directive. When such an error is detected it is normally useless to try to proceed with the code check run - but if the LF stop is by-passed, X13 will continue reading the program thereby enabling any further errors of the same type

to be picked up.

As the last routine of a program is not normally followed by a directive, X13 will not look for missing tags in this routine. Also X13 does not automatically look for missing S-type tags. To enable X13 to carry out both these functions, the order

26 989N

should be punched on the tape after the last routine. This order instructs X13 to clear out the entire directory (this is quite an important feature in itself) and to look for missing S-tags, missing S-parameters, and L-tags missing from the last routine. An LF stop will occur if any of these faults are found. (A missing S-parameter arises when a complete routine is omitted from the program tape.)

It is strongly recommended that this "clear directory" order be performed with every program. A typical tape will end with three "start" orders, viz:

(1)	"Clear directory"	26 989N
(2)	Stop tape	24 999N
(3)	Start program	2B ND.

Whenever an LF stop occurs the contents of the A register (in sexadecimals) should be noted. This is the directory entry referring to the floating address order for which the corresponding tagged order has not been provided. The tape should also be marked to indicate at which point it stopped in the reader -- this will indicate from which routine the L-tag was missing. This information can then be analysed in the manner described in the full write-up.

In general, X13 is suitable for shorter programs, where its length is not a disadvantage (X13 occupies locations 879 to 1023 and uses a "directory" the length of which is under

the control of the user, and which occupies locations immediately preceding 879.)

5.18 METHODS OF ENTERING THE D.O.I. Following is a summary of the effects of entering the D.O.I., X1, at different locations. Unless the contrary is stated, these entry points also apply with X2, X12 and X13.

Transfer control orders on the program tape:

20 1019N: When read from the tape this order causes the input of the program to stop. When SILLIAC is restarted with the black switch it recommences input as if the stopping order had not been on the tape.

24 999N: This order also stops the input of the program. When restarted with the black switch the directive which was being used just before this order was read will be lost, so the next thing read from the tape must be either a new directive or else another transfer control order terminated in N.

Transfer control orders within a program:

26 999F: Used after an interlude to recommence reading of the program. If the first order read from the tape is not a directive the contents of the Q register will be used as a directive, in which case Q must contain nothing else apart from the address at which the new program must be stored.

26 1014F: Has the same effect as 26 999F except that the directive (if not on the tape) is

taken from the A register instead of the Q register (not available with X12, X13).

22 1014F: The number already in location 2 is left there and is added on to all orders terminated by L, but the store location at which the new orders are stored is determined by the contents of the A register. Only the right-hand address digits of the A register are sensed, the contents of the others are irrelevant.