THE 903 ALGOL INTERPRETER

CONTENTS

/Contd.

/Contd.

THE 903 ALGOL INTERPRETER

Part 1: Introduction and Basic Routines.

1.1. Introduction.

The Algol Interpreter is used to run a 903 Algol program in the form of object code produced by the Translator. This description should be read in conjunction with the Elliott 903 Algol Object Code Manual (June 1966).

The Translator reads Algol source text, checks it and converts it into an object code program which consists of parameter words (pords), data and workspace. In the basic Algol system for 8K store, this object code is punched in relocatable binary form on paper tape.

The Interpreter reads the program output by the Translator, assembles it into store, and obeys the object code by interpreting the pords in a manner similar to the computer logic obeying machine code instructions.

The Interpreter consists of:-

1) The Algol Loader (based on the SIR relocatable binary loader).
2) The Pord Evaluation routine, which sets up the initial states and then 'obeys' each pord.
3) A set of subroutines.

   To interpret an individual pord, the Evaluation routine selects one of a large number of sub-rountines according to the code value in the pord. These routines (not necessarily written in standard subroutine form) together form the great bulk of the Interpreter. They are subdivided into 3 classes:-

   a) The Arithmetic subroutines, which act on integer, real and boolean values; performing addition, multiplication, exponentiation, equivalence, etc.

   b) The organisation routines, which perform various tasks such as transferring values to and from the stack, setting up stack entries for procedure calls, making branches in the pord

/Contd.

b) Contd.

program, etc.

c) The input-output routines.

1.2. Initial Addresses.

The store from location 8 to location 123 contains a set of entry points, starting addresses and working locations, whose absolute addresses are required to be known. The position of each item is fixed and must not be moved, as any such change would make it necessary to change the majority of all library and users machine code procedures. Certain locations in this range have been left spare, any further items requiring fixed locations may be allocated to these positions or locations immediately above 123. Any new program or additional instruction in existing routines, must be placed at a higher position in store.

A brief description of the store from locations 0 to 123 is given below. For further details the program coding and the individual program descriptions should be consulted.

Address	
0 to 7	These locations are not referred to explicitly at any point in the Interpreter. They are used only implicitly as the S.C.R. and B registers of whatever level the interpreter is obeyed from, normally level 1.
8	Entry to READAL (see 1.3.1.)
9	Entry to CONTIN (see 4.3.)
10	Entry to EXECUT (see 1.4.)
11	Entry to loader (Start C) to input a relocatable binary tape related to the previous tape input. (see 1.3.2.)
12	Entry to LIBENT (see 1.3.3.)
13	Entry to READOL (see 1.3.4.)

/Contd.

Address	
14 & 15	(Spare in Issue 1)
16	Entry to CHNGOP, which sets +4 in STODEV and then comes to a dynamic stop. This causes the lineprinter to be treated as the presumed output device for future entries to EXECUT.
17	Entry to PUNCH 1, which sets +1 in STODEV and comes to a dynamic stop. This restores the punch as the presumed output device.
18 to 20	(Spare in Issue 1)
21 to 25	are reserved for special entry points or addresses for individual users machine code programs.
26	Labelled STKMOD, this location holds either zero or the address at which the stack is required to start. If zero the stack will start at the first free location given by the loader after all programs and data have been loaded. It holds zero in the first version of Algol. If an extra store module was to be used to hold the stack it might be set to +8192 for example.
27	Labelled STKEND, this location holds the address of the block following the area reserved for the stack. Its current value is +8177.
28	Labelled WARNAD, this location holds an address somewhat lower than the beginning of the Algol Loader. If on entry to a procedure or creation of an array the stack pointer exceeds this address WARN is set non zero. (See 1.3.1. for the effect of this). Note that the stack can grow a short distance beyond WARNAD without setting WARN due to ordinary statements. If the stack is not in the first module WARNAD should be set equal to STKEND.
29	Labelled PDADD, this location is used by the loader as the first address for storing Algol pords. It normally holds the first free location after the Algol Library functions.
30	STACKA, is filled by the loader with the next free location after

/Contd.

Address	
30 (Contd.)	any program or data read in.
31	Is labelled BASE, for access to 32 and 33, and also holds the address of the first free location after the interpreter.
32 & 33	are filled by the loader with the addresses corresponding to QACODL and QAVNDA, the Object Data Load (constants) and the ~~Notimal~~ Data Area (variables), respectively. /Notional
34	WARN (See 28, WARNAD).
35	PP (Pord Pointer) holds the address of the next pord to be obeyed.
36	SP (Stack Pointer) holds the address of the 'top' of the stack, the location pointed to and all higher locations are free.
37	EP (Entry Pointer) holds the address of the stack entry made on entry to the current block.
38	FP (Formal Pointer) holds the address of the result space (followed by parameters, if any) of the current procedure. In issue 2 it will only be set for machine code procedures.
39	BN holds the current block and is always an exact multiple of 16.
40 to 55	hold addresses of various routines that are required by library programs. See the program sheets for details.
56 to 65	are special workspace locations required by library programs to be in fixed locations.
66 to 79	are further program addresses and workspace locations required by library programs, plus some spare locations.
80 to 123	are workspace locations which are used by library programs and which may also be used by users machine code subroutines. They may be used freely by machine code procedures so long as these

/Contd.

Address

80 to 123 (Contd.)

do not call interpreter subroutines. If they do call such subroutines the user should consult the subroutine specifications and the program sheets where necessary.

1.3. Special entry points to the Algol Loader.

1.3.1. READAL.

This routine is the most commonly used entry to the loader. It causes the store pointer to be reset to the first free location after the library, thus preserving the library in store but overwriting previous Algol object code programs. The dictionary pointers are set to preserve the Library dictionary only.

Before entering the Loader WARN is tested. If non-zero a previous Algol program has overwritten the Loader. In this case the message 'RELOAD TAPE 2' is displayed and the program exits to STOP.

1.3.2. Entry at 11 (Input related tape).

This entry goes direct to the loader without testing WARN and does not reset the store or dictionary pointers. It must only be used after an entry at 8 or 13.

1.3.3. LIBENT.

This routine tests WARN as in 1.3.1. above. If WARN is zero the Loader is entered at a point which causes the entire dictionary to be deleted and the store pointer reset to the first free location following the Interpreter. A marker is set (in location 75) to indicate that the current tape supplies the Library. This effectively causes the store pointer and dictionary beginning pointer to be preserved and used for subsequent entries to READAL.

/Contd.

1.3.4. READOL.

This entry is used to input a program which does not need the Library or has the necessary Library functions on its own tape. It performs the same operations as LIBENT (1.3.3) except that the marker to indicate Library input is not set. Thus the Library and Library dictionary are overwritten, but no addresses are stored for future use.

1.4. The Pord Evaluation Routine.

This routine commences at EXECUT which sets up various initial states. If BASE +2 is still +8191 a correct Algol program cannot have been loaded, and the routine exits to STOP.

Otherwise the routine sets up PP, SP etc., and the standard state for input-output. The standard presumed settings are placed in the global settings position, taking as output device number the value in STOPEV. This value is +1 for Punch 1 unless set by entry 16 to CHNGOP.

The routine then 'obeys' the first pord by jumping to NXPORD.

NXPORD is entered from EXECUT and from the end of every pord routine in the interpreter. It picks up the word pointed to by PP, stores the least significant 13 bits in ADPART. It increases PP by one, and decodes the most significant 5 bits of the current pord by jumping to a 32 word lookup table FBAJ. This consists of jumps to the basic interpreter function routines, which decode the value in ADPART according to the individual function characteristics.

In the case of the two functions, PRIM and INOUT the address part specifies one of a further set of sub-routines. INOUT is described under the heading of Input-Output. PRIM is specified further below. All other function descriptions may be found by the reference in Section 3.

PRIM.

The address part of pords with function PRIM specifies one of a group of 70 primitive sub-routines, whose addresses are listed in a table PBA ~~P.B.A.~~ Primitives 1 to 29 are entered by a direct jump to the address

/Contd.

listed. Primitives 30 to 70 are written in standard sub-routine form with the link location at the address listed. Before entry to Primitives 30 to 56 W is set equal to SP-6 and SP is reduced by 3. Before entry to primitives 57 and above W is set equal to SP-3.

Primitives 1 to 29 are described in the 903 Algol Object Code Manual, under their individual names. Primitives 30 to 62 are described in Section 2 (the Arithmetic routines).

Primitives 63 to 70 are used to place a check number next to a parameter address. They use a common routine which places (N-60) in location SP-2, where N is the primitive number.

/Contd.

Part 2

903 Algol Arithmetic Narrative

The Algol arithmetic can be conveniently divided into a number of sub-routines, which are entered directly from the pord-evaluator. These sub-routines are labelled PRIM for primitive and each one has a different number associated with it so that each PRIM sub-routine is unique, e.g. PRIM59. Each sub-routine performs one arithmetic operation on either real numbers or integers, or in the case of the functions e.g. LN. for logarithm, on one number only.

Before entering any of these PRIM sub-routines, the pord evaluator uses a location SP, which holds the current value of the stack pointer, to set a workspace location W which then indicates the position in the stack of the number or numbers to be operated on.

The PRIM sub-routines themselves use a few service routines for organisational purposes and some double length arithmetic.

Section 1 of this narrative describes these service routines briefly and section 2 gives a fairly full account of the PRIM S/R's.

2.1. Service Routines

Section 1

The name of the service routine precedes its description.

SINGLE:	sets a real number from the stack into w/s locations W3, W4, W5.
FAIL:	This is an error S/R which outputs the error number in the accumulator at the time of entry and if continuation is effected sets ± floating point ∞ into [illegible] W3, W4, W5 dependent on the sign of W3.
RSTACK:	Sets 2 real numbers from the stack into w/s locations W3, W4, W5; W6, W7, W8;
RRES:	Sets the real number in W3, W4, W5 back into the stack.
SET:	Sets boolean results into the stack. Entry at SET+1 sets +0 into the stack indicating false. Entry at SET+6 sets +1 into the stack indicating true.
STAND:	Standardises the real number in W3, W4, W5. If the number is too small for representation or zero W3, W4, W5 are all set zero. If the number is too large then an error is indicated using FAIL (as above).
SHIFT1:	Halves the double-length mantissae in W3, W4; W5, W6 and increments their exponents in W5, and W8 respectively. surely W6 W7 ?
DCMULT:	Multiplies together the double-length numbers in W3, W4; W6, W7 and places the double length answer in W3, W4.

ITOR: Changes the integers in W3, W6 to real numbers and stores the standardised results in W3, W4, W5; W6, W7, W8.

CHEBY: Calculates the value of a chebyshev polynomial whose d/ℓ argument is stored in T, T+1.
The number of constants to be used is set in C, and the starting location of these d/ℓ constants, which must be sequential and in order, is set in cstart.
Forms a sequence:
i.e. $B_r = 2 \times arg \times B_{r+1} - B_{r+2} + C_r$

DLDIV: Divides the d/ℓ number in W3, W4 by that in WS17, WS18, and sets the d/ℓ answer in W3, W4.

2.2. Arithmetic Primitives

SECTION 2

The name of the PRIM s/r precedes its description. All real numbers must be in standardised form on entry and are standardised before exit.

PRIM30: Adds 2 integers which are stored in the stack and replaces the first with their sum. If overflow occurs Error 3 is indicated and a dynamic stop obeyed.

PRIM32: Subtracts the second integer in the stack from the first and replaces the first with this difference. Overflow is treated as in PRIM30 above.

PRIM34: Multiplies two integers, held in the stack, and replaces the first with their product. Overflow is as for PRIM30 above.

PRIM38: Raises the first integer in the stack to the power indicated by the second. The result is placed where the first integer was. If overflow occurs error 3 is indicated (see PRIM30 above). If the 2nd integer is -ve then error 20 is indicated and a dynamic stop obeyed. The following results should be noted

$0^X = 0$ for all X

$X^0 = 1 \quad X \neq 0$

PRIM41,43,45,47,49,51:

Examines the two integers in the stack and, depending on their values and the particular primitive, replaces the first primitive with a Boolean result. i.e. +1 for True, and +0 for False. There follows a list of the primitive nos. and their functions.

Prim41	If $I_1 < I_2$ then True false otherwise
" 43	$\leq$
" 45	$=$
" 47	$\neq$
" 49	$>$
" 51	$\geq$

PRIM31: Adds two real numbers held in the stack and replaces the first with their sum. If the answer is too large for representation then error 9 is indicated and if continuation is affected then the answer is assumed to be $\pm$ Fl. Pt. ∞

PRIM33: Subtracts the second real number in the stack from the first and replaces the first with this difference. Overflow is as for Prim31 above.

PRIM35: Multiples together two real numbers which are held in the stack and replaces the first with their product. Overflow is as for PRIM31 above.

PRIM37: Divides the first real number in the stack by the second and replaces the first with the answer. Overflow is as for PRIM31 above. The following results should be noted

X/0 gives overflow provided X $\neq$ 0

0/X = 0 for all X

PRIM42,44,46,48,50,52:

Examines the 2 real numbers in the stack, and, dependent on their values replaces the first with a boolean result +1 for true, +0 for false. There follows a list of the prim nos. and their functions.

PRIM42	if $R_1 < R_2$ then true, false otherwise.
44	$\leq$
46	$=$
48	$\neq$
50	$>$
52	$\geq$

PRIM53,54,55,56,57:

Examines the 2 boolean numbers in the stack and depending on their values and the prim no. replaces the first with a boolean result, +1 for true +0 for false. There follows a list of the prim nos. and their functions.

PRIM53	If B1 AND B2 then true, false otherwise.
54	OR
55	EQUIV
56	IMPLIES
57	NOT B1

PRIM36: Divides the first integer in the stack by the second and replaces the first with the real answer. The S/R first converts the integers to real numbers and then uses PRIM37 (see back) so that errors and special cases are as for PRIM37.

PRIM39: Raises the first integer in the stack to the power indicated by the second and replaces the first with real answer. Overflow is as for PRIM31 (see back). The following results should be noted

$0^X = 0$ for all X

$X^0 = +1.0 \quad X \neq 0$

PRIM40: Raises the first real number in the stack to the power indicated by the second and replaces the first with this result. The subroutine uses PRIM61 (log) and EXP (exponential) so that errors which occur in log and exp will also occur here. The following results should be noted:

$0^X = 0$ for all X

$X^0 = +1.0 \quad X \neq 0$

PRIM58: Replaces the real number in the stack by its absolute value.

PRIM59: Replaces the real number in the stack by its entier (an integer). If the result is too large for integer representation then error no. 3 is indicated as in PRIM30 (see back).

PRIM62: Replaces the real number in the stack by
+1 if it is +ve.
+0 if it is zero.
-1 if it is -ve.

PRIM61: Replaces the real number in the stack by its Logarithm. If the number is -ve or zero then error no. 13 is indicated and on continuation the result is assumed to be zero.

EXP: Replaces the real number in the stack by its exponential value. If the real no. if zero the answer is given as +1 immediately. If the real number is >+40 then error 12 is indicated and on continuation the result is assumed to be $\pm$ Ft. Pt. ∞ Results that are either too large or too small for representation are treated as in PRIM31.

DIV: Divides the first integer in the stack by the second and replaces the first with the integer result, rounded towards zero if necessary. If the second integer is zero or the result is too large to store as an integer then error no. 3 is indicated as in PRIM30. The ws location W is not set by the pord evaluator in this routine so that on entry the routine uses SP (stack pointer) to set W before obeying the division.

Part 3: The Organisation Routines.

These routines are adequately described in the 903 Algol Object Code Manual under their individual names. For further details the flow diagrams and program listing should be studied. Sub-routines used by the Organisation routines are described in Part. 4. INOUT and the input-output routines are described in Part.5.

/Contd.

Part 4: Subroutines used by the Organisation routines

4.1. FAILEN.

This subroutine prints information when a run time error is found by any interpreter routine.

Entry: Place link in PAUSRT, jump to FAILEN with an error number in the accumulator as a positive interger.

Exit: Exit is standard with W and W3 holding the same value as on entry.

Process: The routine displays the readings:-

ERROR BN PP RETURN followed on a new line by the given error number, the current block name and pord pointer, and the address in the second word of the stack entry for the current block. ~~Each is printed as an integer prefixed by *.~~ The error display is preceded by output of 16 blanks on the punch.

After output the routine comes to a program wait. Since the link is in PAUSRT re-entry at 9 causes exit from the subroutine (see CONTIN 4.3.). If the error is one from which recovery is not possible the subroutine entry should be followed by a stop.

Workspace: The global and local ~~point~~ pread sellings are left unchanged by this routine, W3 is reset to its initial value. W1, W2 and W4 to W10, SBW to SBW5; WS and ADPART are all left undefined.

4.2. ENFAIL.

Entry is by direct jump to ENFAIL with an error number in the accumulator. A standard entry is made to FAILEN followed by a stop. This entry is used for non recoverable errors. Another entry is ERROR which is used for compiler errors, e.g. incompatible data. SPARE is entered from all spare positions in the primitive and pord function tables (representing another compiler error).

/Contd.

4.3. CONTIN.

This is entered from the program entry address 9. Its entry parameter is the line in PAUSRT, which is always set before the interpreter enters a program Wait. A jump is made to one plus the address in PAUSRT.

Before exit PAUSRT as set to cause STOP on re-entry, so that if the computer should stop for reason other than a program Wait (e.g. output device in Manual) re-entry at 9 will have no effact. Note that the program can always be restarted by entry at 10.

4.4. STSUBRT (Assign).

This subroutine assigns a given real or integer value to a given address, with optional packing if the value is real.

Entry: Store the link in STSUBRT.

Enter at:-

1) STSUBRT +1 with the address at (SP) -6 and the value at (SP) -3.

2) STSUBRT +7 with the address at (PKDADD) and the value at the address held in the accumulator.

3) STSUBRT +15 with the address at (PKDADD) and the value in W3, W4, and W5.

Exit: Exit is standard if entry (1) is used.

SP: = SP-3 and W: = SP-6

Process: If the given address is that of a constant (bit 17=1) a non-recoverable error indication is given. If the address is that of a real value (bit 18=1) the given value in 3 locations is packed into 2 locations, with rounding to the nearest 2^{-27}. If the exponent is less than $^{-64}$ zero is assigned, if greater than $^{+63}$ the error for floating point overflow is given and $\neq$ 0.99999999* 2^{63} is assigned. If the location following the address is negative, the number is assigned without packing

4.5. W34SR.

This subroutine copies a real value (in unpacked form) from a given

/Contd.

position into locations W3, W4 and W5.

Entry: Store link in SBLNK and use entry:-

1) W345R copies the topmost value in the stack. a
2) W345R1 copies the value whose address is held in the accumulator.
3) W34R2 as for (2) with address in B register.

Exit: Standard.

Workspace: W3, W4, W5. Also W if entry (1) or (2) used.

4.6. RICONV.

The subroutine converts the real number in W3, W4, W5 to integer form. The process used gives the Algol automatic type conversion (integer: = enter (real + 0.5)). i/

Entry: Store line in SBLNK1 and enter at RICONV.

Error: If the number is greater than 131071.5 or less than -131071.5 the error for integer overflow is given. Continuation is not possible.

Exit: Standard, with the integer in the Accumulator and W3.

Workspace: W3, W4, W5, SBW.

4.7. ITRSB1.

This subroutine converts an integer to a real value. The real value is in standardised floating point form, unpacked in locations W3, W4 and W5.

Entry: Store link in SBLNK, enter at:

1) ITRSB1 with the integer in the Accumulator.
2) ITRSB2 with the integer in W3.

Exit: Standard.

Workspace: W3, W4, W5, SBW.

/Contd.

4.8. NEGR1.

This subroutine negates a given real value in standard unpacked floating point form.

Entry: Store link in SBLNK, enter at:-

1) NEGR1 with the address of the real value in W.
2) NEGR6 with the address in the B register.

Exit: Standard, with the negated number in the original position.

Workspace: Entry (1) W. Entry (2) none.

4.9. FINDFP.

This subroutine finds the formal pointer corresponding to a given block number.

Entry: Standard subroutine entry, place link in FINDFP and enter at FINDFP +1. ADPART contains the given block number BN^{1} and n (the procedure parameter number) in the least significant 4 bits.

Exit: Exit is standard, with 3n + FP^{l} in the Accumulator, where FP$^{\ell}$ is the formal pointer corresponding to BN$^{\ell}$.

Errors: If the given block name is not found a "compiler error " is printed (recovery not possible).

Workspace: SUBWK1, SBW1.

4.10. GARAD.

This subroutine gets the address of an array map, given the address of the array.

Entry: Store link in SBLNK and enter at GARAD with the address in the Accumulator.

Exit: Standard, with the array address in W and the map address in W1.

Workspace: W, W1.

/Contd.

4.11. AINDSB.

This subroutine finds the absolute address of an array element, given the indices in the stack. It is used by INDA and INDR pord functions.

Entry:	Standard, place link in AINDSB and enter at AINDSB +1. ADPART hold~~s~~ n, the number of dimensions, and the indices are held in stack at the positions SP -3n, SP -3n + 3,....., SP-3. The array address is at SP-3n-3.
Exit:	Standard, with the element address in the Accumulator. If a real array, bit 20 has value 1.
Error:	If the array element falls outside the limits of the array a non-recoverable error is given. (Array index wrong).

/Contd.

Part 5: The Input-Output Routines.

5.1. INOUT.

This routine is used to call a further set of subroutines, depending on the value in the address part of the pord. It determines whether the Inout number is greater than 15 to inform the setting procedures that a local or global setting is required. The address of the particular Inout routine required is obtained from a table at INOUT 3. All the routines are entered as subroutines with a common link at IOLNK. However, only numbers 1 to 4 make use of IOLNK on exit, all the others exit direct to NXPORD.

5.2. Character input and output.

All input and output via paper tape station is done through the subroutines GECHAR or OTCHA. (This includes output to the teleprinter and line printer where fitted.). The only exceptions to this rule are the output of blanks to the punch in FINISH and FAILEN and the input of tape by the Algol Loader.

5.2.1. OTCHA.

This subroutine outputs one character to the current output device. It is identical in the 920 version to the standard 903 version.

Entry: Store link in OTCHA and enter at OTCHA 1 (OTCHA +1), with the character to be output held in the Accumulator and the required output device number held in ODEV.

The character to be output is either:-

(a) A six-bit 903 Internal code character.

(b) A binary pattern to be output, with the sign bit set to one.

Entry to OTCHA2 causes a repetition of the previously output character.

/Contd.

X

Exit: Exit is standard, with ODEV unaltered.

Workspace: SBW.

Method: If the sign bit is not present, the output code value of the character is found from TABLE. If a double character is indicated, the character in VBARCH is output before the translated character. (This is redundant in the 903 version). The output is done by using ODEVTB [ODEV] to modify a/15 0 instruction.

If the character is internal code newline, the character from TABLE is followed by output of the character from LFCH and a blank. (In the 920/503 version LFCH is blank also).

A test is included for output device numbers greater than 4. At present this merely causes a jump to exit from the subroutine. At a later date this may be used to enter a digital plotter output routine and/or other special output device routines.

5.2.2. GECHAR.

This subroutine effectively gets one character from the current input device. Different versions are provided for the 903 and (920/503 code) Interpreters. Entry and exit are identical for both versions.

Entry: IDEV must contain the required input device number.

Enter at GECH with the link stored in SBLNK.

or Enter at GECHEN with the link stored in GECHAR.

Exit: Exit is standard. The character just read from tape and translated to internal code is held

/Contd

in BUFFER [IDEV].

The previous character (in internal code) is held in NEXTCH. The group code for this character is held in the Accumulator on Exit. (For a description of groupcode see the Translator description, page 21). (For the 920 Interpreter substitute "character just taken from the input buffer" for "character just read from tape").

Method: (1) 903 Interpreter.

A character is taken from BUFFER (IDEV) and stored in NEXTCH. If this character was marked with the sign bit it represented a halt code and a wait stop is entered. (The initial routine EXECUT ensures that 'space' is the first character taken at the start of a program).

The program then reads one character from the current input device. A/15 0 instruction is modified by IDEVTB [IDEV]. Seven bits of this character are used to look up TABLE values. The parity bit is compared with that from the table. The least significant 6 bits of the table look up give the internal code value, except for special characters. Special characters are grouped into: illegal characters, ignorable characters, newline and halt.

The appropriate internal code is stored in BUFFER [IDEV]. The groupcode TABLE [NEXTCH] is taken before exit from the subroutine.

Method: (2) 920 Interpreter.

A character is taken from BUFFER (IDEV) and stored in NEXTCH. BUFLAG is then tested. If

/Contd.

negative the line input buffer is empty. (EXECUT sets BUFLAG negative at the start of each program).

If the line buffer is not empty the next character is picked up and stored in BUFFER [IDEV]. The groupcode for NEXTCH is found by looking up TABLE and the routine exits. If the character picked up from the line buffer was newline then BUFLAG is set negative and HALTMK tested. If HALTMK is <u>true</u> the newline character represents a halt code input and S^WAIT (Systems wait) is entered.

If the line buffer is empty characters are read from paper tape, converted to internal code, and packed 3 to a word until the next newline is read. Vertical bar is treated as a special character, the following significant character is read and the appropriate internal code formed. Halt code is treated in the same way as newline, with a special setting of HALTMK to <u>true</u>. If an illegal character is found the buffer if filled up to newline before an error is given.

When the line buffer is full the routine returns to pick up the first character from the buffer and exit in the normal way.

5.3. Output number routine.

Function:	To print a real number or integer on a specified output device, using a specified 903 Algol format.
Entry:	In all cases the link is stored in IOLNK.

1) Enter at OUTR to print the real number at (W) using the current local print settings. A holds th

/Contd.

new value of SP.

2) Enter at OUTI to print the integer at (W) using the current local print settings. A holds the new value of SP.

3) Enter at OUTR2 with the required print mode given in the Accumulator to print the real number in (W). The output device number must be set in ODEV and the required digit settings in DIGM and DIGN where appropriate:

Output mode	Accumulator	DIGM	DIGN
Freepoint (n)	+ 0	(not used)	+n
Aligned (m,n)	- 1	+m	+n
Scaled (n)	+ 1	(not used)	+n

4) Enter at OUTI4 to print the integer held in the accumulator. ODEV must hold the output device number required. The current digits number is taken from INTDG.

5) Enter at OUTI5 to print the integer held in W3. ODEV must be set as for (4) and the accumulator must hold +n for a print in the style: digits (n).

Exit: Exit is to INOEX, which causes a standard subroutine exit via the link in IOLNK.

<u>Method:</u> Integers are converted by the standardise routine and printed as real numbers in the form aligned (n,0). The real number is converted from the form:

$N = a * 2^b * 10^0$ where $-1.0 \leqslant a < -0.5$ or $+0.5 \leqslant a < +1.0$

to:

$N = F * 2^0 * 10^x * SIGN$

where $+1.0 > F \geqslant +0.1$

X is an integer

SIGN is +1 or -1

/Contd.

The modulus of N is first taken, then X is found by repeated multiplication or division by +10.0, until F is in the required range when b is reduced to zero.

If the number cannot be printed in the specified format an alarm print takes place according to the rules in the 903 Algol Manual. Otherwise the number is printed by OTMXD as a mixed number. If in scaled format the number is printed as F * 10^1 and followed by subscript 10 and (X-1) printed as an integer with sign and non-significant zeros.

Working locations: W, ODEV, are not altered by the routine. W3, W4, W5, DIGM, DIGN, SBLNK, SBW, SBW1, SBW2, ADPART, SBLNK2, WS2 to WS8, WS14, NSIGNF, SIGNCH, TENPWR, SW, W1, W2, W7, W8, W9, WS10, WS11, WS12, are altered in an undefined manner.

5.4. Input number routine.

Function: To input a number punched in one of the standard Algol forms from the current input device.

Entry: Store link in IOLNK and enter at RDNM. To read real number ADPART must be set +2, otherwise an integer is assumed. W points to the address to which the number is assigned.

Exit: The number is assigned to the given address. SP is set equal to W. The number may be found in W3, W4 and W5 (W3 for integer). Exit is via INØEX which causes a standard return ~~to~~ via the link in IOLNK.

Method: The routine always reads a real number, converting to integer form before exit if required. Each digit input is mul~~tiplied by 10~~ scaled and added to a double length mantissa, which is multiplied by 10.0 before the addition.

/Contd.

A count is kept of the number of digits after the decimal point and this is combined with the decimal exponent (read as an integer). The floating point number formed from the digit input routine is then multiplied or divided by 10.0 the appropriate number of times.

Before exit the number is negated if a negative sign was read, and converted to integer form if required.

5.5. Other INOUT routines.

These routines take their parameters from the top of the run-time stack. For read and point settings one routine is used for both global and local settings, the difference having been detected by INOUT. These routines detect whether the settings are out of range, and where necessary replace them by the standard settings. Thus the read and point routines do not have to allow for out-of-range settings.

For further details the 903 Algol Object code Manual, the flow diagrams and program sheets should be studied.

/Contd.