SCHOOL OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

POP-2/4100 USERS' MANUAL

By

Raymond D. Dunn

February, 1970
Revised July, 1972
Incorporating Newsletters 1 - 6

CONTENTS

INTRODUCTION

This manual describes the multi- and uni-access versions of the conversational programming language POP-2 implemented on the ICL 4100 computer series. Certain sections are only relevant to one or other of these systems, known respectively as Multi-POP and Uni-POP, and are clearly marked as such.

The systems described are those in use in the School of Artificial Intelligence; certain facilities may differ, or be excluded, in POP-2/4100 systems available at other installations. Details will be found in the documentation relating to specific installations.

POP-2

POP-2 is a conversational programming language developed by R.M. Burstall and R.J. Popplestone in the Department of Machine Intelligence, University of Edinburgh, and is described in Programming in POP-2 (1971). A working knowledge of POP-2 is assumed in this manual.

Certain facilities described in the Reference Manual are not available, or are implemented differently, in the POP-2/4100 systems. It is recommended that reference should be made to this manual before any facilities are used.

Multi-POP

Multi-POP is a multi-access version of POP-2 based on the ICL 4130. The hardware configuration on which the system is currently running in the School of Artificial Intelligence, is as follows :-

4130 processor with 64K of 2μs core	(32K)
control teleprinter	(1)
2 paper tape readers	(1)
2 paper tape punches	(1)
8 channel teleprinter controller	(4 chan.)
1 real time clock	(1)
1 300 LPM lineprinter	(-)
3 4 million character disc drives	(-)
2 H316 computers, one of them connected to the Department's experimental robot	(-)

(The figures given in brackets indicate the minimum hardware requirements for the system.)

A maximum of eight users may use the system simultaneously and are all permanently core resident, sharing the available I/O facilities. Time-sharing between users is by a simple 'round-robin' mechanism, all users having equal priority.

Uni-POP

Uni-POP is a uni-access version of the above which can be run in one of two modes :-

(a) As a dedicated single user conversational programming system using the control teletype, or

(b) As a dedicated POP-2 batch processing system using paper-tape input and lineprinter output.

The minimum hardware configuration is as above without the multi-teleprinter controller and real time clock.

Errors and changes to specification

The correction of any errors, additions, or changes to the specification of any facility described in this manual will be published in the POP-2/4100 Newsletters which appear from time to time. Copies may be obtained from the address below, to which all enquiries should also be made :-

D.J.S. Pullin,
School of Artificial Intelligence,
University of Edinburgh,
Forrest Hill,
Edinburgh,
EH1 2QL.

Telephone : 031-226-3101 Ext. : 29

THE TELETYPE AND ITS USE

The console used is a 4100 coded teletype. POP-2 identifiers, and POP-2 systems words which are normally underlined, are both typed in upper case without underlines, thus systems words such as VARS, FUNCTION, END, etc. may not be declared as identifiers.

The standard POP-2 character set is defined in APPENDIX 2. If any character is typed other than those listed, or whose effect is explained below, it will normally be ignored, and if it is a printing character, a back arrow (←) will be output to signify this.

The dialect of the ISO code specified for the Edinburgh Regional Computing Centre may be used with the following substitutions :-

for	½	type	£
"	´ (acute accent)	"	' (apostrophe)
"	10	"	?
"	` (grave accent)	"	@
"	£	"	~
"	←	"	␣ (underline)

Four other characters are used as special control characters in the system; these are produced using CTRL with another key, and cause ↑ followed by the relevant letter to appear on the console. They are :-

CTRL and G (called BELL)
CTRL and O
CTRL and R (called READY)
CTRL and T (called HALT)

Note when the SHFT and CTRL keys are used, they should be pressed down first and held down while the other required key is pressed - do not press both keys simultaneously.

Input buffer

All characters typed on the teletype are stored in a software 'buffer' and are passed to the system as complete lines. A line is defined as zero or more characters followed by the RETURN key. The system acknowledges this by outputting a LINE-FEED. It is possible to continue over more than one physical line on the teletype by typing LINE-FEED immediately followed by RETURN, but note that the maximum number of characters the system will accept in one line is 128; (the effect of typing the 128th character is the same as if CR had been typed, i.e. the line is terminated).

Note that LINE-FEED followed by RETURN is passed to the system as a single character.

When HALT is typed, the character TERMIN (see section on Input/Output) is passed to the system and the line is also terminated. This can only be used in certain documented cases, and will otherwise have an undefined result.

The system indicates when it is ready to accept input by outputting either a colon (:) or two colons (::) followed by a space, and after typing CR the user must wait for this to be output before typing anything further. Any characters typed by the user before colon and space appear are ignored, and ← is output by the system after each such character.

Character editing

Because of the buffering, it is possible to edit lines of characters before they are passed to the system. This is achieved by the use of the ← and ! characters.

- ← deletes the immediately preceding character on that line, successive use deleting the corresponding number of preceding characters. Note that space is treated as a character in this context. If ← is typed as the first character on a line it has no effect.

- ! deletes the whole line up to it; the system acknowledges the fact by outputting CR/LF followed by colon and space, and waits for the line to be retyped.

NOTE. The above facilities may be used up to the typing of RETURN, but as soon as RETURN is typed, the line is passed to the system as it stands.

Parity checking

All characters are checked for even parity as they are typed; if an odd parity character is detected, it is ignored, and the system outputs ← to signify this. Excessive rejection of characters in this way indicates a hardware fault in the teletype in use, or a bad line to the computing system. The latter may often be cleared by reinitiating the connection.

Length of output lines

The teletype is set to have a maximum line length of 72 characters. When outputting, to avoid overprinting at the end of a line, the system substitutes a new line for the first space encountered after the 65th character has been printed. If no space is found, a new line is inserted immediately after the 72nd character has been printed.

Example of teletype input

```
: 2+3←4 =>                    (replace 3 with 4)

** 6,
: SORT(9) +!                  (delete the line)
: SQRT(9) + 5 =>
A←                            (character typed too soon)
** 8.0,
: ^ABC EF←←←DEF^ =>           (note that space is significant)

** ^ABCDEF^,
```

Important restriction

As the input buffer is also used by the system when outputting characters, if any output occurs before a line has been completely processed, the rest of the line is lost.

Thus :-

```
: 3+2 => 5*3 =>

** 5,
```

when the first => is encountered, it is obeyed, and the resulting output causes the remainder of the line to be lost.

This is only troublesome when typing statements in execute mode, i.e. outside a function body, as no output can take place while a function is being input.

Particular care should be taken to note the effects of this, especially with respect to COMMENTS (see page 23).

Automatic stop/start reader facilities

It is possible to input paper-tapes directly from teletypes fitted with automatic readers (within School of A.I., all converted

teletypes (i.e. those with 4 lights/buttons on their right hand front) have this facility).

The mechanics of the reader work as follows :-

The control lever is biased towards the centre. In this position it can be controlled automatically by the computer. If the lever is pushed upwards, this starts the reader manually, and if pushed downwards a small amount, this stops the reader manually.

In the fully down position, the tape is free to be moved backwards and forwards. When using the automatic facilities, the lever should always be returned to the centre position.

A standard function POPAUTOREAD (ɛtruthvalue=>()) is provided to enable the user to set his console into automatic reader mode (POPAUTOREAD(TRUE)) or to return to normal mode (POPAUTOREAD(FALSE)).

To have a tape prepared in the manner given below read from the teletype automatically, the user should load it in the reader, leave the control in the central position and type POPAUTOREAD(TRUE);CR. The tape will now be input.

When in automatic reader mode a line of input is terminated by the X-OFF (CTRL S) character, and RETURN is completely ignored. X-OFF also has the effect of switching the tape reader off. This process takes several characters to be fully accomplished, and so X-OFF on tapes should be followed by at least 3 blank (run-out) characters. When the system is ready to accept more input, as well as outputting colon/space, it outputs the X-ON (CTRL Q) character which switches the reader back on again.

Thus tapes prepared off-line for input to Multi-POP on teletype readers should be typed with X-OFF followed by at least 3 blanks at the end of each 'buffer' of input. A 'buffer' can be up to 128 characters long, but users will probably find it convenient to follow each CR/LF by X-ON/BLANK/BLANK/BLANK except where lines are very short.

Backarrow (←) and exclamation mark (!) may be used to edit tapes being prepared off-line in a similar way to their use when typing on the teletype, however, it should be noted that ! causes all characters before it up to the last X-OFF to be ignored, and CR/LF is not output.

If a parity error is detected when reading in automatic mode, the whole buffer which included the error is ignored, the system outputs : ! on a new-line, and the reader is not switched back on.

The user can thus type this buffer in again remembering to terminate it with the X-OFF character, when the rest of the tape will now be read automatically. Similarly if the tape contains more than 128 characters before an X-OFF, the whole section from the previous X-OFF is ignored and the system reacts in the same way as for a parity error.

The HALT character may be used on tapes in the same way as it would be used when typing except that ↑T is not output and HALT does not terminate the buffer.

These differences between typing and reading from the reader are essential to ensure that the system and the reader are always in step with each other.

Normally all tapes prepared for use in this way should be terminated by: POPAUTOREAD(FALSE); CR/LF/X-OFF/RUNOUT. This will cause the system to revert back to normal mode and the reader to remain off. If a whole session's input was prepared on tape it could be terminated by: POPAUTOREAD(FALSE); .LOGOFF; CR/LF/X-OFF/RUNOUT and thus the user could set his console into automatic mode when logging on and leave it to complete his run and log him off. This should be particularly useful for long production jobs.

Output to console in automatic reader-mode : To enable users to output files in a form suitable for input on the teletype reader, the system, when in automatic reader mode, follows every CR/LF output to the teletype by X-OFF/BLANK/BLANK/BLANK. Tapes thus produced may be re-input on the teletype reader with no difficulties.

Errors : If an error which calls the standard system error routine occurs, the teletype is automatically switched back to normal mode.

SETPOP does not reset the the teletype to normal mode.

Suppression of teletype output

It is often useful to be able to stop and start output appearing on the teletype without affecting the process which is producing this output (e.g. to stop the monitor printout produced by the command ON in POPEDIT without affecting the edit process, when one has forgotten to do an OFF command).

This facility is provided by use of the CTRL p key.

a) If the console is in the process of printing:

The CTRL P key should be pressed, and then pressed again within ½ sec. while the output pauses.

If this is successful, ↑P will be printed and the output will immediately stop (as if an assignment equivalent to ERASE -> CHAROUT; had been done).

b) If the console is in input mode pressing the CTRL P key has no effect.

c) If the console is in wait mode (i.e. neither in input mode nor outputting) pressing the CTRL P key causes ↑P to be printed and will inhibit any output occurring.

Output is restored on the console by pressing the CTRL P key a second time and also after anything is input on the console.

CTRL P may be used to inhibit any output on the console, no matter what its source. (i.e. from user programs, systems messages etc.)

Example

```
: FUNCTION F; LO: A.PR; A+1 -> A; GOTO LO END;
: .A;
1 2 3 4 5 6 7↑P↑P                    (CTRL P pressed and then pressed
                                      again after a short time.)

254 255 256↑P↑P                      (Output restarts - process has
                                      been continuing without output.
                                      repeated.)

1567 1568 1569↑G                     (CTRL G pressed to abandon
                                      process.)

SETPOP:
:
. FF();

COMMENTS [FF↑P:                      (CTRL P inhibits the rest of
: .LOGOFF;                            the comments message and all
                                      the error 37 that is given.)

CPU TIME US↑P                        (Stop rest of logoff message.)
```

Automatic Logoff

Incautious use of the CANCEL facility can make it impossible for a user to logoff in the normal fashion (see below). To remedy this the user should hold down CTRL and SHIFT and type K.

This will transfer control to the LOGOFF routine and the normal message will appear.

It can also be used if a private crunch has occurred under Multi-POP 70.

If the teletype is outputting, the key will need to be pressed twice, as described in the previous section.

LOGGING IN TO THE MULTI-POP SYSTEM

After the user has set up his connection to a free channel of the system (information on which can be found in the relevant section of the folder 'Multi-POP Information for Remote Consoles' which

accompanies all Multi-POP consoles), he should switch his teletype to the LINE position. (For teletypes with a SIMPLEX/DUPLEX switch, DUPLEX should be selected and the ON button pressed).

The system is activated by pressing BELL, all other input being ignored.

MULTI-POP SYSTEM. ISSUE <issue>. <time> <date>.

is output, where <issue> is the issue number of the system in use, and <time> and <date> are the real time and date of the run. A system message may then be output, and the demand :-

NAME:

appears. The user should type in his identifying initials (followed of course, by the CR key). If an acceptable name is not given, the demand is re-output, otherwise the message :-

STORE <n> BLOCKS FREE:

appears, where <n> is an integer and denotes the number of blocks of 512 words of store currently available for use. The user should type in the number of blocks he wishes to reserve (0 to <n>) which will normally have been pre-arranged through the system booking scheme.

If a reply is unacceptable to the system, i.e. an integer in the range 0 to <n> is not input, the store request message will be re-output. Note that the number of blocks of store available may change between the store request being printed and the user typing a reply.

When an acceptable number of blocks have been requested, the message :-

DISC TRACK:

is output. The user can type one of three options :-

i)	0	if he does not wish to use the standard disc filing system, or has not yet got a track initialised.
ii)	an integer <n>	which causes the files on track <n> to be available.
iii)	a list of integers	which causes the files on all the given track numbers to be available, and the head of the list to be the current track.

(For full details see page 41).

Then the message :-

SETPOP: (see page 25)

is output on the teletype and the system is ready for use.

If a user wishes to return to the start of the logon process before the disc request is output, he should type the BELL character.

LOGGING OFF THE MULTI-POP SYSTEM

When the user has finished with the system, he should log off by typing :-

LOGOFF();

The system will output the message :-

CPU TIME USED = <time used>.

LOGOFF <time> <date>.

where <time used> is the amount of actual central processor time used during the run, to the nearest sixteenth of a second, and <time> and <date> are the real time and date at the finish of the run.

On logging off, the user's workspace is cleared, and all devices currently allocated to him are closed. Care should be taken to ensure that any disc files in use have been terminated before logging off. (See page 41).

LOGGING INTO, AND OFF, THE UNI-POP SYSTEM

When the system has been input and set up (see APPENDIX 4), the demand :-

NAME:

appears. The user should type in his identifying initials followed by carriage-return, line-feed,

SETPOP:

is output and the system is ready for use. The user's name is required in Uni-POP both for file identification and for disc use.

To log off the system if another user is about to use it,

```
LOGOFF();
```

should be typed, the store will be cleared, and the demand :-

```
NAME:
```

will be output, ready for the next user.

THE MULTI-POP STORAGE RESERVATION SCHEME

When a user logs on to the Multi-POP system, the store he reserves is not immediately allocated to him, and the user is not strictly limited to staying within the limit of this reservation. When a user's program requires store, the system attempts to allocate it for him, and if none is free, a scan is made to determine if any user is exceeding his reservation. If such a user is found, he is logged off by the system, the message :-

```
** CORE REQUIRED = <n> BLOCKS
```

being printed, where <n> is the number of blocks being used. If more than one user is exceeding his reserved amount, the worst offender is 'killed' by the system first. Note, however, that 'joyriders' (users who are logged on with a reservation of zero blocks), are killed before any others.

If, after the above has been carried out, the required amount of store still cannot be found, the user requiring it is given error 50, the CULPRIT being the actual number of words of store which the user required.

A standard variable, COREUSED, holds the number of words of store currently being used by a user. This variable is automatically updated by the system every time a garbage collection takes place, and may be used to determine how many blocks should be reserved for each job.

Changing Multi-POP store reservation

At any time during a run, the user may change the amount of core currently reserved for his use by applying the function STORE.

STORE(<n>); changes the user's reservation to <n> blocks.

The function gives a boolean result indicating whether or not the store requested was available, e.g.

```
: STORE(10) =>

** 1,          indicates that the user has successfully
                changed his reservation to 10 blocks.

: STORE(30) =>

** 0,          indicates that 30 blocks are not available,
                the user's reservation remains unaltered.
```

Clearing store

A standard function, CLEARPOP, is provided to clear all workspace currently in use; the dictionary is cleared, the stack is reset, all devices allocated to the user are closed, and the system asks for the user's new disc requirement.

```
: CLEARPOP();

SYSTEM CLEARED. RESTART
DISC TRACK:
```

PROGRAM INTERRUPT FACILITIES

Full interrupt facilities which allow either continuation or the abandoning of the current process are implemented and have been so designed that unintentional interruption, either by the user or by poor data transmission lines, is almost impossible.

In Uni-POP

Pressing the message key will cause an interrupt. If the console is in the process of printing, the current buffer will be output before the interrupt takes effect. SETPOP is then entered as for CTRL G under Multi-POP.

During a batch run, the effect is slightly different - see APPENDIX 4.

In Multi-POP

A process is interrupted by pressing either the CTRL G or the CTRL R keys to give either an immediate call of SETPOP (thus abandoning the job) or an exit to READY level respectively. This is done by :-

a) If the console is in the process of printing:

The required key should be pressed, and then pressed again within $\frac{3}{4}$ second while the output pauses.

If this is successful then ↑G or ↑R will be printed, otherwise output will continue and the user should try again.

b) If the console is not outputting:

The required key should be pressed once. This will be printed as ↑G or ↑R.

If CTRL G is used, the system will now immediately call SETPOP (q.v. page 25 and all current processes will be abandoned. All variables retain their current values, and may therefore be interrogated.

If CTRL R is used, the teletype will finish outputting its current output buffer (up to 128 characters), if any. The system then enters the user-definable function POPRDYFN of no parameters, which is set initially to be IDENTFN. On normal exit from this function, the system outputs

READY

and waits for input from the console. When at READY level the system indicates a request for input by outputting 2 colons (::) followed by a space to remind the user that he is at this level.

If POPRDYFN requires input from the console, the message

READY::

is output at that time instead of later.

If "GOON" is compiled inside POPRDYFN, the suspended process is re-started as described below.

The printing of READY signifies the following :-

i) The current process has been suspended.

ii) The stack is 'frozen' at the state it was in at the time of interruption (its contents are <u>not</u> available to the user).

iii) All variables are left with their current values.

iv) Input and output revert to the teletype and ERRFUN is given its standard value.

The user is now free to start any other process (e.g. he may wish to change the values of variables, redefine function definitions, turn bugging on or off, etc., or do something completely unrelated to the suspended process). This process itself may be suspended by READY or the last suspended process re-started by typing GOON. This resets the system exactly as it was at the time of suspension except that all variables redefined during the suspension keep their new values.

The number of processes which can be suspended simultaneously is entirely dependent on the stack used by each process.

A standard function POPREADY(ε()=>()) is available to the user. When called, this has the same effect as if CTRL R had been pressed.

THE STACK

The operation of POP-2 is based on a stack mechanism. This is a last on-first off storage mechanism, and because of its importance in the system, a short explanation of how it is used, both explicitly and implicitly, by the user is given here.

Without exception, the values of variables, elements of data structures, arguments and results of functions etc., are all accessed and assigned to via the stack. Consider the statement :-

```
X;
```

This can be interpreted as 'load the value of the variable X onto the stack'.

Further, the assignment statement :-

```
X->Y;
```

can be read as 'load the value of the variable X onto the stack; remove the top item from the stack and store it in the variable Y'. Thus the assignment statement :-

```
->Y;
```

with no apparent source is meaningful, and the statement :-

```
X, Y->X->Y;
```

will swap the values of the variables X and Y.

On entry to a function, the values for the formal parameters are taken from the stack, and any results of the function are left on the stack, this can be seen by the following two functions which are exactly equivalent :-

(1)
```
FUNCTION LOGADD X Y => Z;
  LOG(X) + LOG(Y) -> Z;
END;
```

(2)
```
FUNCTION LOGADD;
VARS X Y Z;
  -> Y -> X;
  LOG(X) + LOG(Y) -> Z;
  Z
END;
```

The use of the variable Z is, of course, unnecessary, and is only included to make the point clear, the following function (3) being the most common form of doing the above,

(3)
```
FUNCTION LOGADD X Y;
  LOG(X) + LOG(Y);
END;
```

It should be noted that any possible combination of the methods used in these functions is meaningful, and is legal POP-2.

Finally, it follows from the above, that when a function is called, its arguments must be supplied on the stack, thus :-

```
        (a)  LOGADD(A,B)
and     (b)  A; LOGADD(B);
and     (c)  A; B; LOGADD();
```

are all equivalent in their effect; they all provide two arguments on the stack for the function LOGADD. Functions with n arguments may apparently then be supplied with less than n (e.g. in (b) above), so long as all the required arguments have been put on the stack before the function is called. Similarly, more than n arguments may be supplied, the last n being taken by the function.

Understanding and full use of the stack leads to simplification and flexibility when programming. The stack may be used to hold any number of values (subject to the restrictions given below), either temporarily or semi-permanently.

The effective length of the stack is fixed at 330 locations, and is used to hold function entry information as well as its normal program use. This information is stored at the opposite end of the stack from that used by the user, is put there on entering a function body and is removed on exit from the function. It consists of a two word link (giving information about the position the function was called from) and the saved values of all variables local to that function.

The amount of space required is 2+n locations of the stack for every entry of a function with n local variables (including formal parameters), and thus the depth of functions to which a program can go (including recursion) is limited.

This depth is dependent on the number of local variables of the functions entered and on the amount of stack used by the user himself. It can be calculated for a particular program from the figures given.

If an attempt is made either to remove something from the stack when it is empty, or to add something to the stack when it is full, an error is given (errors 52 and 51 respectively). For reasons of efficiency, this checking is done by noting the position of the 'stack pointer' whenever the system executes a function entry, a function exit, or a backward GOTO. If the 'stack pointer' is outside the limits of the stack when these mechanisms are obeyed, the error is given, and the culprit is FNENTRY, FNEXIT or BACKJUMP respectively. Because several stack operations may take place between checking points, it is possible for the stack to go out of bounds and return to a legal state without the system spotting the error, the result being undefined.

Undetected overflow cannot occur unless more than 20 items are put on the stack between checks, but undetected underflow can be caused by only one stack unload-load operation.

Common causes of stack overflow and underflow

The most common cause of stack underflow is not providing enough arguments for a function, and of overflow is an infinitely recursing function or group of functions.

Note that the compiler and systems routines all use the stack, and a stack overflow can occur during their execution. Common instances where trouble is met are listed below :-

a) COMPILE

If this is used recursively to more than a depth of 3 or 4, stack overflow may occur.

b) DATALIST

This function puts all of the components of the data structure onto the stack before creating a list. The maximum length of structure it will be able to handle depends on the stack usage at that particular time, but will be less than 320 in any case.

c) CHARACTER STRINGS

When the system reads a string, the stack is used to hold the characters (3 to a location) until the end of the string is read. This then limits the length of strings to be about 900 characters at the most. However, because the stack usage is undefinable by the user while systems routines are in operation, it is wise to keep their length to below 250 characters.
NOTE. Because of this, if a closing string quotes is omitted, the compiler will continue reading program as part of the string, and error 51 will eventually be given.

d) LIST EXPRESSIONS

Each element of the list is put on the stack before the list is created, thus list expressions producing large lists may give trouble.

Resetting the stack

The stack is cleared by the function SETPOP (see page 25).

ARITHMETIC

The 4130 has a 24-bit word, and for normal operations both integers and reals are held in one machine word. A set of functions is provided to operate on 48-bit real numbers, using the hardware floating point facilities. These are described at the end of this section. Because the POP system requires one or two bits for control purposes the specification of numbers is as follows :-

Integers lie within the range $-2^{21} \leq I \leq 2^{21}$ i.e. from -2097152 to 2097151. Reals are held as a 6-bit exponent and 17-bit mantissa, giving 5 decimal digits accuracy. Reals must lie in the range $-1*2^{31} \leq R \leq (1-2^{-16})*2^{31}$, i.e. from -2147000000 to 2147000000. Because the fifth digit of accuracy is quickly lost during arithmetic operations, only 4 decimal digits are shown when reals are printed. The smallest fraction which can be represented is 2^{-33} i.e. approximately 10^{-10}.

High precision reals have a 9-bit exponent and 39 bit mantissa giving 10 to 12 digits accuracy. The range is from $-4.3*10^{76}$ to $5.8*10^{76}$, the smallest number being approximately $4.3*10^{-78}$.

Arithmetic is handled interpretively, i.e. the type of the arguments (real or integer) are determined at run time, the following operations being available :-

Operation	Precedence	Explanation	Arguments	Result
<	7	less than	Integer or real	Truthvalue
>	7	greater than	Integer or real	Truthvalue
=<	7	less than or equal	Integer or real	Truthvalue
>=	7	greater than or equal	Integer or real	Truthvalue
+	5	add	Integer or real	Integer or real
-	5	subtract	Integer or real	Integer or real
*	4	multiply	Integer or real	Integer or real
/	4	divide	Integer or real	Real
//	4	integer divide with remainder	Integer	Integer
↑	3	exponent	Integer or real	Real

+, - and * produce an integer result if both arguments are integer, otherwise a real.

// produces two integer results, the remainder and the quotient in that order.

If an operation on two integers gives a result outside the range of integers, the number will automatically be converted to type real.

ERROR 45 is given if any of these operations (or other arithmetic function) is applied to a high precision real.

The other arithmetic functions provided are :-

Function	Explanation	Argument	Result
INTOF	Entier (Integer part)	Real	Integer
REALOF	Convert to real	Integer	Real
SQRT	Square Root	Integer or real	Real
SIGN	-1, 0, or +1 according to sign	Integer or Real	Integer
SIN	Trig function	Integer or real (In Radians)	Real
COS	Trig function	Integer or real (In Radians)	Real
TAN	Trig function	Integer or real (In Radians)	Real
LOG	Natural logarithm	Integer or real	Real
EXP	Natural anti-logarithm	Integer or real	Real
ISNUMBER	Test for number	Any item	Truthvalue
ISREAL	Test for real	Any item	Truthvalue
ISINTEGER	Test for integer	Any item	Truthvalue

Note that INTOF produces the integer part and not the nearest integer as specified in the Reference Manual.

Restriction

The representation of zero is the same for both integer and real, and thus a type change may be experienced in a variable if its value was real and passed through zero.

High precision reals

A set of standard functions is provided to allow double-length (48-bit) floating point operations to be performed.

As these items require 2 words of core per number, a storage allocation would normally be required after each operation, making them slow and inefficient. Because of this, the facilities have been kept separate from the normal arithmetic operations, and have been implemented as 'pseudo machine-code' instructions in which each user has his own floating point accumulator (FPA)

In the following specifications,

FPA - is the users' floating point accumulator.
simple - is a standard POP-2 integer or real.
hpreal - is a record containing a 48-bit real.
It has no selectors or updaters, and has
the dataword "HPREAL".
number - is either a simple or a hpreal.

HPROF ε simple => hpreal. Does not affect the FPA.
HPRLD ε number => loads the FPA.
HPRADD, HPRSUB, HPRMUL, HPRDIV
all ε hpreal => the respective operation between
the FPA and the hpreal to the FPA.
HPRLT, HPRGT, HPREQ, HPRLEQ, HPRGEQ all ε hpreal => truthvalue
i.e. compares the given hpreal with the FPA and gives
truthvalue for the respective operations corresponding
to <, >, =, =<, >=.
These operations do not affect the FPA.
HPRUNLD ε () => hpreal
i.e. forms a HPREAL record from the FPA.
Does not affect the FPA.
HPRST ε hpreal => puts the FPA into the given hpreal (i.e. a non-
constructive unload).
Does not affect the FPA.
HPRRLOF ε hpreal => real
i.e. converts a hpreal to a real.

Notes:

i) An hpreal is never equal to a simple.
ii) An hpreal is a compound item.
iii) Hpreals cannot be input or output directly, the functions HPROF, HPRLD and HPRRLOF must be used.
iv) No type checking is done except in the case of HPRST. This further improves efficiency.
v) The functions are much faster than their equivalent real operations.
vi) HPRRLOF will give ERROR 33 is an attempt is made to convert an hpreal which is too large to represent by a real. It will give zero for hpreals too small to represent by a real.

Example of use

Let us first define some operations and a macro to simplify the use of these instructions :

```
VARS OPERATION 1 (++ -- ** /// && --> >>);
```

```
HPRGT -> NONOP>>;
HPRADD -> NONOP++;
HPRSUB -> NONOP--;
HPRMUL -> NONOP**;
HPRDIV -> NONOP///;
HPRLD -> NONOP&&;
HPRST -> NONOP-->;
MACRO ->>; MACRESULTS([;.HPRUNLD->]) END;
```

Note that ->> causes a 'constructive' assignment, i.e. an HPREAL record is formed, whereas --> causes the FPA to be put into an existing HPREAL record.

We can now do for example :

```
: &&0 ->> A;              (load the FPA with 0, create an HPREAL from
                           the FPA and assign it to A)
:
: ->> B;                  (The above does not affect the FPA, this produces
                           another HPREAL of 0 and assigns it to B)
: &&1.56 ->> C ->> D;     (Load the FPA with the high precision value of
                           1.56, create 2 HPREAL records of it and assign
                           to C and D)
: D =>
**  <HPREAL>,             (D is an HPREAL record)

: D.HPRRLOF =>            (Print its real value)

**  1.56,
: HPRRLOF(7.5)-> E;       (Create an HPREAL of 7.5 and assign to E)
: --E --> A;              (Subtract E from the FPA and put the result
: A.HPRRLOF =>             in A without creating a new record)
**  -6.06,                (Print the real value of A)
: >>E =>                  (Do a test equivalent to FPA>E)

**  0,                    (False, i.e. -6.06<7.5)
```

If we now want to do the equivalent of the arithmetic expression: (A + B) * (C - D/2) -> E; we proceed as follows, remembering that these are equivalent to machine code instructions, and no rules of precedence can exist. (Assume A, B, C, D and E are hpreals as above).

```
: ->> WS;                 (Create an hpreal from the FPA (contents not
                           relevant) and assign to WS as a workspace
                           record)
: &&2 ->> TWO;            (Load the FPA with 2 and create an hpreal
                           assigning it to TWO)
: &&D///TWO --> WS;       (Load the FPA with D and divide it by TWO
                           non-constructively storing this in WS)
: &&C--WS --> WS;         (Load the FPA with C, subtract D/2 from it and
                           store back in WS)
: &&A ++ B ** WS --> E;   (Load A, add B, multiply by C-D/2 and store the
                           result in the hpreal E)
```

ITEMS AS BIT-STRINGS

All the functions defined in the Reference Manual to operate on bit-strings are implemented as standard. They may be applied to any item, but when they are, that item will be regarded as a positive integer with a maximum size of $2^{22}-1$. Bits 23 and 24 will be set to zero.

INITIAL VALUES OF VARIABLES

When a variable is declared, the system initialises its value to be a pair whose FRONT is the name of the variable, and whose BACK is the word UNDEF. Macros and operations are initialised to a call of ERROR 37. Thus :-

```
: VARS TEST;
: TEST =>

** [TEST. UNDEF],
```

INITIALISATION OF FUNCTION NAMES AND USE OF FNPROPS

The FNPROPS of a function is used by the system to hold the name of a function, and is also used to hold the SPEC information for the debugging routines (see page 56). When a user wishes to use the FNPROPS for his own purposes, he should, to avoid inconsistencies, take this into account, and leave the FNPROPS in a format which can be handled by the systems routines. The format of FNPROPS is initialised by the system to be as follows :-

a) With anonymous functions FNPROPS is set to be NIL.

b) With functions which are input in the form :-

```
FUNCTION <name>...;

  :
END;
```

The FNPROPS is set to be a list whose head is <name>.

When a function is SPEC'ed the FNPROPS is set to be a list whose head is a list in the form :-

```
[<name>[<parameter names>][<result names>]]
```

Thus when the system wishes to access the name of a function (e.g. to print it), it assumes that it is either the head of the FNPROPS or if this is a list, then the head of the head.

The user should thus always use the tail of the FNPROPS, and if this is NIL, a list whose head is NIL (or a name defined by the user) should be created when inserting user information.

SECTIONS

Sections have been implemented differently from the description given in the Reference Manual. The discrepancies are :-

i) Sections must be given a name.

ii) The external list may be divided in two by the identifier =>. Only those elements which follow the => are included in the macro expansion of the section name.

iii) Users' identifiers in the text enclosing the section may not be used inside the section unless they appear in the external list.

iv) Syntax words and standard identifiers which cannot be re-defined by the user are made internal to each section; those which can be re-defined are treated as though they appeared in the first part of the external list.

v) Users' identifiers which are used inside the section without appearing in the external list, are internal to the section, and cannot be accessed outside it.

COMMENTS

Whenever the compiler encounters an identifier which it has not met before (i.e. has not been declared), it is added to the user's dictionary as a new name, global for the current SECTION. If the standard variable POPCOMMENT is set to TRUE, the name is added to a list of such names, and as soon as the end of the current statement, if in execute mode, or the end of the current function, has been reached, the word COMMENTS followed by this list will be output to the teletype. If POPCOMMENT is false, no output occurs.

ERRORS

On the detection of an error, either at compile time or run time, the system calls the standard function ERRFUN. This is normally defined to output a message of the form :-

```
ERROR <number>
IN FUNCTION <name>
IN SECTION <name>
CULPRIT <offender>
```

If the standard variable POPDOTRACE is set to TRUE the function POPTRACE is entered, which outputs :-

```
CALLING SEQUENCE:-
FUNCTION -
FUNCTION -
:
FUNCTION POPVAL
```

POPAUTOREAD(FALSE) is obeyed and control reverts to the teletype by a call of SETPOP.

<number>	is the number of the error condition.
<name>	is the name of the function and section(s) (if any) in which the error was detected, and
<offender>	is the illegal item met. The actual significance of this depends on the error given, and is normally self-evident.

A full list of error numbers with their meanings can be found in APPENDIX 3 of this manual.

In some cases it is not possible for the system to output either one, or both, of the function name and culprit (e.g. if arithmetic overflow occurs, the offending number is already too large to handle, and cannot be printed).

POPTRACE can also be called by the user at any time and will output the calling sequence. The last function shown in the trace will always be POPVAL (i.e. the compiler); all user functions currently active (including anonymous ones) will be shown as will relevant active systems functions.

ERRFUN may be redefined by the user. It is given two arguments, the culprit and the error number, in that order, and may be defined to exit by the call of a function produced by JUMPOUT (see Ref. Manual). If a user defined value of ERRFUN is allowed to exit normally (i.e. through its end) when called by the system, the function SETPOP will be entered. This restriction is required because the system routines are not designed to be able to continue after an error condition has been met.

The user may call either the standard value or his own value of ERRFUN from his program. In this case, the function SETPOP is not entered.

A call of SETPOP resets ERRFUN to its standard value.

SETPOP

When called, the function SETPOP has the following effect :-

a) The current process is abandoned.

b) The stack is cleared.

c) All active SECTIONs are exited from.

d) All barriers set up by BARRIERAPPLY are abandoned, and subsequent attempts to REINSTATE any previously created state will cause an error.

e) CUCHAROUT, the current output device, is reset to CHAROUT, i.e. output reverts to the teletype.

f) ERRFUN is reset to its standard value.

g) DEBSP is set to zero (see page 58).

h) Any names on the COMMENTS list are output.

i) The word SETPOP is output on a new-line.

j) Input reverts to the teletype and PROGLIST is reinitialized.

Note that the current values of local variables are not affected, and that SETPOP does not alter any device allocation. However, because CUCHAROUT is reset, users should take care that character repeaters are not lost (see page 38).

INPUT/OUTPUT FACILITIES

Files and character repeaters

Information may be supplied to the POP-2/4100 system from files, and similarly the user may output information to files. Associated with each file is a device.

The basic facility supplied for handling such files is the character repeater.

An input character repeater is a function of no arguments, which when applied, produces as its result the next character from its associated file.

An output character repeater is a function which, when given a character as its argument, outputs that character to its associated file.

To make this clear, consider the on-line teletype. 'Files' associated with the teletype are known as the Standard Input File, and the Standard Output File. Two functions are supplied for transferring characters to and from these files :-

CHARIN	is the input character repeater for the teletype.
CHAROUT	is the output character repeater for the teletype.

Thus the instruction CHARIN()->X; will cause the system to read a character from the teletype and assign its value to the variable X.

Similarly, CHAROUT(33); will cause the character A to be output to the teletype. (NOTE: characters are handled as integers - their internal code representation, see APPENDIX 2). e.g. the word FOX could be output to the teletype by the sequence: CHAROUT(38); CHAROUT(47); CHAROUT(56); (easier methods of doing this will be discussed later).

Before a file associated with any device other than the teletype can be accessed, it must be opened, after which characters may be transferred one at a time until the file is closed. Every file has a name, which is any list of words and/or numbers.

The standard function POPMESS is used for controlling file operations. To open a file, POPMESS is supplied with one argument, a list whose head is the device name and whose tail is the name of the required file; its result is the required character repeater.

The device names available are :-

1. PTIN For paper tape input (ISO code). Each tape character is decoded to POP-2 internal code as it is read.

2. PTOUT For paper tape output (ISO code). Each character is encoded from the POP-2 internal code to ISO code.

3. PTBININ For binary paper tape input. Each character is read as an 8-bit binary number. 8-, 7- and 5-hole tapes may all be read by this facility.

4. PTBINOUT For binary paper tape output. Each character is output as an 8-bit binary number.

NOTE that there are two readers and two punches available, so that each of the above two input and output facilities is available on both readers and both punches respectively.

5. LP80 For printing to the lineprinter. A new line is automatically inserted after the 80th column.

6. LP120 For printing to the lineprinter. A new line is automatically inserted after the 120th column.

7. ROBOT For communicating with the experimental robot.

8. BIO316 For communicating with the Honeywell H316 in the Bionics Laboratory.

9. OPERATOR For sending messages to the computer control-teletype.

For example, the following two lines open a paper tape input file called [PROG DATA] and a lineprinter output file called [PROG RESULTS], and assign the character repeaters produced to the variables DATAIN and RESOUT respectively.

```
POPMESS([PTIN PROG DATA])->DATAIN;
POPMESS([LP120 PROG RESULTS])->RESOUT;
```

A file may be closed in one of several ways. There is a standard variable, TERMIN, which when output as a character, closes the appropriate file. In the case of input files, reading the terminating character causes the file to be closed, and the result TERMIN to be given by the repeater. In this way the user can easily determine when the end of a file has been reached. In the example below, the input file is closed when its terminating character is read by IN; TERMIN is assigned to X which is then output by OUT, closing the output file; the TERMIN is detected and an exit is made from the function.

Files may also be closed at any time by applying POPMESS to a list of two items, the first being the word CLOSE, and the second being either the character repeater for the appropriate file, or the identifier which holds the repeater, e.g. if CR is a character repeater for a file, it can be closed by POPMESS([%"CLOSE",CR%]); or POPMESS([CLOSE CR]);. This is useful when a user does not wish to read a complete file - it can be closed at any time.

Three further facilities exist under POPMESS :-

i) POPMESS([CLOSEALL]);

Closes all devices currently associated with the user.

ii) POPMESS([STATUS <device name>]);

Obtains the status of the device. The results are :

```
    0  -  device available
    1  -  device in use
UNDEF  -  device unknown to the system.
```

For instance,

```
L1: IF POPMESS([STATUS LP80])
      THEN GOTO L1 CLOSE;
```

would wait for the lineprinter to become available, but should not be done because

iii) if a device is in use when an attempt is made to open it, the message

```
WAITING FOR <device>
```

is printed and the user is swapped out until the device becomes free, unless the current user is also the previous user, when it is immediately re-assigned.

CTRL G or CTRL R can of course be used to do something else while the device is busy.

When the user is swapped back in, the message

```
<device> OPENED
```

is printed.

Examples

To copy a file from any input device to any output device, we can write a function such as :-

```
: FUNCTION TRANSFER IN OUT;
: VARS X;
:  LO:
:   IN()->X; OUT(X);
:   IF X = TERMIN THEN EXIT;
:   GOTO LO
: END;
```

this can be used for example to print up a paper tape file, by doing :-

```
: TRANSFER(POPMESS([PTIN<name 1>]),POPMESS([LP80<name 2>]));
```

where <name 1> is the name of the paper tape input file and <name 2> is the required name of the output file.

It could also be used for punching a paper tape from input typed on the teletype by, for example :-

```
: TRANSFER(CHARIN,POPMESS([PTOUT TEST 1]);
```

this gives the output file the name [TEST 1], which will contain all the characters typed in on the console, being terminated when the HALT key is pressed (giving the character TERMIN).

Any number of character repeaters may be used simultaneously (the limit being the number of devices available), and, for example, output can be got on several devices. The line :-

```
IN()->X; OUT1(X); OUT2(X);
```

copies a character from one input device to two output devices etc. and could be used in the function TRANSFER above for this purpose.

Time limits on devices

A user is normally only allowed to use a device for a maximum of five minutes. If a longer or shorter time is needed, this can be specified by replacing the device name in the POPMESS statement by a list of two items, the devicename and the time in minutes for which the device is required.

```
e.g.   POPMESS([[LP80 20] TEST 1])->CR;
```

opens a lineprinter file with the name [TEST 1], giving a maximum time of 20 minutes for its use.

After the time limit has expired, the device is not closed for use by the user unless another (or the same) user calls for that device, in which case the original file is closed and any further use of its character repeater will result in an error being given.

Use of specific devices and the format of files

PTIN

This is used for inputting ISO-coded paper tapes, each character being decoded to its POP-2 equivalent.

The tape must begin with its name, which is any list of words and/or numbers, e.g. [TEST1],[PROGRAM FILE] etc., and must be terminated by a HALT character (STOP-CODE on Flexowriters). On tapes produced on off-line equipment, the character following the HALT should be blank tape or erase, as tapes produced by the system have a check sum following the HALT (see PTOUT).

All characters, including carriage-return, erase, blank tape and lower case letters, which are not included in the POP-2 character set, will be ignored on input.

Files are opened on this device by using POPMESS in the way described previously, and are closed either by using the CLOSE facility of POPMESS, or by reading up to and including the HALT character on the tape. The file will be closed by the system and an error message given if an odd parity character is detected, if the checksum fails (see PTOUT), or if the reader is unloaded while the file is being read.

PTOUT

This is used for outputting ISO-coded paper tapes, each character being encoded from POP-2 to ISO-code on output. Line feed is output as RETURN followed by LINE-FEED.

The characters output are preceded by a list which is the name given to the file in the POPMESS statement, and when the file is closed, a HALT character followed by a two character sumcheck for the tape is output followed by a length of blank tape. The first character of this check is the sum of all characters on the tape including the HALT, and is punched as six bits with correct parity, the zero sum being punched as bits 7 and 8. The second character is a character count, again being punched as six bits with correct parity.

When tapes produced by PTOUT are input under PTIN, these check characters are read and compared against their computed values, any discrepancy causing an error message. If a tape which fails to check is still required to be read, the character just after the HALT should be erased with a uni-punch. When either erase or blank-tape immediately follows the HALT, no checking takes place on input.

Files are opened on this device by using POPMESS in the standard way, and are closed either by using the CLOSE facility of POPMESS, or by outputting the character TERMIN. The file will be closed by the system and an error message given if the punch runs out of paper tape.

PTBININ

This is used for inputting any paper tape as 8-bit binary numbers, i.e. the character is input as read, no decoding taking place. 8-, 7- and 5-hole tapes may be input by this method, and in the case of 7- and 5-hole tapes, the unused positions are read as binary ones, except the position corresponding to the edge of the tape, which is undefined.

No file name is required on the tape, or in the POPMESS statement, but if given in the latter will aid the operator in loading the correct tape. The file must be closed using the CLOSE facility of POPMESS.

PTBINOUT

This is used for outputting characters as 8-bit binary numbers, i.e. the bottom eight bits of the item given to the character repeater is punched on the tape. Files are opened in the standard way, but here again no file name is punched on the tape, and can be omitted from the POPMESS statement. The file must be closed using the CLOSE facility of POPMESS.

LP80

This is used for outputting files to the lineprinter when 80 character width paper is in use, this being the standard paper size in the School of Artificial Intelligence. A newline is automatically inserted by the system after the 80th character has been output to a line.

The rest of the specification is as LP120.

LP120

This is used for outputting files to the lineprinter when 120 character width paper is in use. This will not normally be available

during a standard Multi-POP session in the School of Artificial Intelligence. A new line is automatically inserted by the system after the 120th character has been output to a line.

Files output to the lineprinter start at the top of a new page and are preceded by the real-time and date when the file was output, the user's identifying initials, and the name of the file as given in the POPMESS statement. For example, if a file is opened with the name [RESULTS 1] the first page of output would be preceded by :-

```
15.30HRS. 10 JAN 1970                **RDD**
[RESULTS 1]
```

where RDD would be the user's identifying initials.

Two special facilities are provided with the lineprinter :-

If the integer 64 is output as a character, the lineprinter will print the next line of output at the top of a new page. Normally a new page is selected after every 60th line has been printed.

If any negative number is output as a character, this has the effect of carriage-return without line-feed and can be used for over-printing of lines.

Files are opened on this device by using POPMESS in the standard way, and are closed either by using the CLOSE facility of POPMESS, or by outputting the character TERMIN. The file will be closed by the system and an error message given if the printer goes into a non-recoverable error state - e.g. 'NO PAPER' or 'YOKE OPEN' etc., 'PAPER LOW' is detected, and a suitable message is output to the operator to load more paper.

ROBOT

This device is used for controlling the on-line experimental robot under development in the School of Artificial Intelligence. Its use is described in the documentation of the program [LIB LINK EXEC] which can be found in the folder 'POP-2 PROGRAM LIBRARY' which accompanies all Multi-POP consoles.

BIO316

This device is identical with the Robot; except that it uses BIO316STATUS as its doublet.

OPERATOR

This provides a facility for transmitting messages to the computer operators via the control-console.

e.g. POPMESS([OPERATOR 'MY CONSOLE IS ON FIRE']);

will cause the given string to be printed on the console in the computer room. The POPMESS leaves no results (although the Fire Brigade may be called).

THE DISC SYSTEM

The system uses 3 4 million character random access disc units. These behave like one disc, divided up into 300 tracks, numbered from 0 to 299; each track has 160 sectors numbered from 0 to 159. A sector contains 256 characters.

Each user of the system is allocated one or more tracks for his individual use, and although he may read information from any part of the disc, he may only write to those tracks allocated to him Each file must start at the beginning of a sector and can thus be regarded as occupying an integral number of sectors.

Functions are available to treat disc files as character repeaters similar to those of POPMESS and to manipulate them in various ways. These are described in Disc Filing (page 41).

Functions to treat disc files as compound structures are described in the library program [LIB EASYFILE STRUCTURES] whose documentation can be found in the folder 'POP-2 PROGRAM LIBRARY'.

For users who will wish to carry out operations not provided by disc filing or [LIB EASYFILE STRUCTURES], the basic facilities are described here.

POPMESS is not used for disc files. Instead, two functions DISCIN and DISCOUT are provided for opening input and output character files respectively. These functions take two arguments, namely the track number and sector number of the starting position of the file on the disc; their result is the corresponding character repeater.

e.g. to obtain a character repeater to output a file starting at track 25 sector 10, one would do :-

```
: DISCOUT(25,10)->CROUT;
```

where CROUT would then be the required character repeater, and the file would extend from sector 10 upwards. Note that a file must

terminate before the end of a track is reached, continuation onto the next track does not take place, and an error is given if this is attempted.

Characters are stored on the disc in their POP-2 form, and output files are closed by outputting TERMIN in the normal way. TERMIN is stored on the disc as decimal 19.

It is important to note that a 256 character buffer is used for disc transfers. The information is only written up to the disc immediately the 256th character (or multiple thereof), or TERMIN, is output, i.e. every complete sector. Thus if a file is not terminated, up to 255 characters may not have been written to the disc. This should be remembered particularly if a process involving disc transfers is interrupted for any reason.

In a similar way, to input a file starting at the same position on the disc, the character repeater is obtained by :-

```
: DISCIN(25,10)->CRIN;
```

The file is closed when the input repeater reads the TERMIN at the end of the file, and again a 256 character buffer is used in the input process, each sector being input to store when its first character is read.

Because of the buffering involved, it is possible to edit a file on the disc in situ - reading the file and outputting it again to the same place in a different form - so long as the editing process does not mean that the new file is more than 255 characters longer than the old, at any time during the process.

Two functions exist to provide facilities for transferring complete structures to and from the disc :-

```
DTOSTRUCT ε TR, SECT, W, STRUCT => W, SECT
STRUCTTOD ε TR, SECT, W, STRUCT => W, SECT
```

where TR is the track for transfer
SECT is the starting sector for transfer
W is a starting offset within this sector in words. (64 words to a sector). Thus if W = 8 then the transfer starts 8 words from the beginning of sector SECT
STRUCT is the structure to be transferred to the disc for STRUCTTOD, or the structure into which the transfer from disc is to go for DTOSTRUCT. STRUCT can be any STRIP, RECORD or ARRAY (of any dimensionality) whose components are not compound (i.e. they can be numbers or items of size <22).

and the results W and SECT give the position on the disc immediately following the block transferred.

One further function is provided - DSECTOR. This function takes a disc character repeater as its argument, and produces as its result the number of the sector currently referred to by the repeater. If it is called immediately after TERMIN has been input or output, this will be the next available sector. In this way then the user can determine the extent of his files on the disc.

Examples

As the disc uses character repeaters just as the other input/output devices do, its use is exactly the same as them.

e.g. again using a version of the general purpose function TRANSFER introduced on page 29.

```
: FUNCTION DTRANS IN OUT;
: VARS X;
:  LO:
:   IN()->X; OUT(X);
:   IF X = TERMIN THEN DSECTOR(OUT) => EXIT;
:   GOTO LO;
: END;
```

In this case we have inserted a call of DSECTOR, so that when the file is closed, the number of the next free sector after the file will be printed.

The following statement allows a user to type a file from his console to disc, starting at track <t>, sector <s> :-

```
DTRANS(CHARIN,DISCOUT(<t>,<s>));
```

as before, the file is terminated by pressing the HALT key.

Similarly, to input a paper tape file [FILE] and write it to the disc :-

```
DTRANS(POPMESS([PTIN FILE]), DISCOUT(<t>,<s>));
```

or to output a disc file to a paper-tape file [FILE] :-

```
TRANSFER(DISCIN(<t>,<s>),POPMESS([PTOUT FILE]));
```

or to type out a disc file to the console :-

```
TRANSFER(DISCIN(<t>,<s>),CHAROUT);
```

In the last two examples we have reverted back to the original function TRANSFER, because of our use of DSECTOR with the output function in DTRANS.

Remember that any number of input/output operations may be combined, and any number of files to, or from, the disc may be open at any one time, although care must obviously be taken to ensure that input files are not inadvertently overwritten by output files.

Input/Output of Text Items

Text Item repeaters may be created from character repeaters by applying the functions INCHARITEM and OUTCHARITEM to them, for input and output respectively.

Thus we may read Text Items from a file whose character repeater is INC, by defining a function, say INITEM, by :-

```
INCHARITEM(INC)->INITEM;
```

Successive applications of INITEM will now produce the sequence of text items from the file, just as the use of INC would produce the sequence of characters.

Similarly, if OUTC is an output character repeater for a file then we may define OUTITEM by :-

```
OUTCHARITEM(OUTC)->OUTITEM;
```

so that OUTITEM("FOX") will output the characters of the word FOX to the file through the character repeater OUTC etc.

When an input item repeater is applied, the characters read by the character repeater are formed into an item under the rules given in sections 9.1 and 2 in the Reference Manual. It is important to note, however, that it is necessary to read the character immediately following each item as it is read to determine whether it is part of the current item. This character is stored in the workspace of the item repeater and is used when the item repeater is next applied. This can create difficulties if the character repeater is used between calls of the item repeater, as this character will still be taken as the next one by the item repeater even though the character repeater has stepped the input on by several characters. Also if this character is in fact not required, it can not be deleted (e.g. in a conversational type program where a user types a reply terminated by some character other than space or new-line - this character would be taken as the first character of the next reply).

The former difficulty may be alleviated by always separating items by spaces in files used in this way, and the latter can be overcome by recreating the item repeater whenever it is wished to particularly avoid troubles with terminating characters. When an item repeater is created, this 'next character' is initialised to be a space, which is ignored at the start of an item.

As an example of this, consider a file, character repeater INC, containing the characters :-

```
99ALPHA;BETA
```

and consider the following operations on this file :-

```
: INCHARITEM(INC)->INITEM;
: INITEM()=>

** 99,                        (produces the number 99)
: INC()=>

** 44,                        (produces the character L,
                               the A having been read and
                               stored in the buffer of the
                               item repeater)
: INITEM()=>

** APHA,                      (Note that the A is used and
                               that the L has already been
                               read from the file)
```

At this point, the item repeater is holding the semi-colon character in its 'buffer', and will produce this as its next item, however, if we now create a new item repeater, the semi-colon will be lost, i.e. :-

```
: INCHARITEM(INC)->INITEM;
: INITEM() =>

** BETA,
```

PROGLIST and input to the compiler

The compiler takes its input from the text item list PROGLIST. This is initially set to be the dynamic list :-

```
FNTOLIST(INCHARITEM(CHARIN));
```

i.e. input is taken from the teletype, but may be set by the user for compilation from other sources (either directly, or indirectly using POPVAL or COMPILE (see page 40).

Three functions are provided to allow the user to read from this list :-

a) ITEMREAD inputs the next text item from PROGLIST.

b) NUMBERREAD inputs the next text item from PROGLIST and checks that it is a number, otherwise it gives an error. Numbers may be signed.

c) LISTREAD inputs the next list constant from PROGLIST. An error is given if anything other than a list is encountered. Any number of sub-lists may be given within the list.

Standard outputting facilities

The system provides several general and special purpose outputting functions which are given below. All these functions use the variable CUCHAROUT (CUrrent CHARacter OUTput device) to determine to which file characters are to be output. The device associated with the value of CUCHAROUT is known as the current output device.

CUCHAROUT is initially set to be CHAROUT (i.e. the console), and is reset to this by SETPOP, but can be assigned any value by the user. Thus the lineprinter can be specified as the device to which output is to be routed merely by assigning the character repeater obtained from a POPMESS statement, to CUCHAROUT, e.g. :-

```
POPMESS([LP80 RESULTS FILE])->LP;
LP -> CUCHAROUT;
```

It is always safer to use an intermediary variable (e.g. LP in the above) rather than assigning directly to CUCHAROUT, otherwise if a call of SETPOP occurs the character repeater will be lost.

It should be noted that CUCHAROUT can take as its value any function which has one argument and produces no results, allowing processes other than outputting to be applied merely by 'printing' items (e.g. it is possible to 'print' characters into data structures or to create complex formatting routines in this way).

Format of items when output

The format of specific items when printed by any output repeater or general purpose printing function is as follows :-

a) Integers

are output preceded by one space if positive, or a minus sign if negative, with no decimal point. e.g. 50, -123, 12379.

b) Reals

are output preceded by one space if positive, or a minus sign if negative. The number is rounded to four significant figures and the decimal point is output with at least one digit before and after it. e.g. -5.023, 754300.0, 0.001.

c) Words

The characters (up to a maximum of eight) are output with no preceding or following spaces.

d) Strings

Are output with their opening and closing string quotes.

e) Lists

The complete list is output, including all sublists, with elements separated by spaces. The list brackets are output. e.g. [1 2 3[[4][5 6]]7].

f) Pairs

If a pair does not contain NIL as its BACK and is not a dynamic link (when it is output as a normal list) then it is output in the format :-

[<front> . <back>]

g) Functions

are output in the following format :-

FUNCTION <name>

where <name> is the name as given in the FNPROPS (see page 22).

h) General data structures

A general data structure is output as its DATAWORD enclosed by the characters < and >. e.g.

<PRECORD>

Functions available

a) PR(X) outputs the item X to the current output device. It produces no results.

b) PRINT(X) outputs the item X to the current output device leaving X as its result.

c) => If used in execute mode, causes every item on the stack to be printed to the current output device, the item on the top of the stack being output last. The output is preceded by two new-lines and **; each item is terminated by a comma. The stack is cleared.

If used inside a function definition it causes the top of the stack to be output on a new line to the current output device, preceded by **. The item is removed from the stack.

=> can be regarded as an operation, with an implied semi-colon before and after it.

d) PRREAL(X,N,M) causes the number X to be output on the current output device with N figures before the point and M figures after the point. If M=0 then the decimal point is not output. The number is preceded by one space if positive, and by a minus sign if negative. If X is an integer then REALOF(X) will be output. If the number is too large to be output in the format specified, it will be output followed by a decimal exponent.

e.g. PRREAL(125.6,4,3);
125.600
PRREAL(720.9,2,4);
72.0900_{10}+01

e) PRSTRING(N) outputs a string without the string quotes.

f) SP(N) outputs INTOF(N) spaces to the current output device.

g) NL(N) outputs INTOF(N) new-lines to the current output device.

COMPILING PROGRAM FILES

There is a special function, COMPILE, for use with files. It is used for compiling a file of source program, and is applied to the character repeater of the file.

e.g. assume a paper-tape program file with the name [TEST PROG] is to be compiled. This can be accomplished by :-

```
: POPMESS([PTIN TEST PROG])->IN;
: COMPILE(IN);
```

This is equivalent to :-

```
POPVAL(FNTOLIST(INCHARITEM(IN))<>[GOON]);
```

but is implemented in a more efficient manner.

When the file has been compiled, it is closed, and control returns to the teletype.

The text of the file to be compiled must not end with an uncompleted imperative expression, function definition, lambda expression or list expression. If it does, the result will be undefined.

The flexibility of character repeaters can again be shown by the following example :-

Assume that a lineprinter listing of [TEST PROG] is required simultaneously with the program being compiled. We can do this by :-

```
: FUNCTION CR;
:   IN()->X; OUT(X); X
: END;
: POPMESS([PTIN TEST PROG])->IN;
: POPMESS([LP80 TEST PROG])->OUT;
: COMPILE(CR);
```

It can be seen that the function CR is an input character repeater which has the side effect of outputting characters by the function OUT. Again any number of simultaneous input/output operations are possible by combining character repeaters in this way, and files may be edited simultaneously with compilation and output (see section on File Editing).

A further function, CARRYON, is provided for use when compiling. This is used to restart the compiling process after a syntax error has been given. The effect of applying CARRYON to a character repeater is that the file is read and copied to the teletype until the word END is read, when COMPILE is again applied to the repeater. This then, allows recovery from compilation errors and also provides a monitor to the teletype of the section of program immediately following the position of the error. (Note that if an error is detected outside a function body, or on reading its END, it is more sensible to use COMPILE to restart; if CARRYON is used, the code up to the next END - i.e. a complete function definition will be copied and ignored).

DISC FILING

A set of standard functions exists to assist with the manipulation of character files on disc.

Initialising disc tracks

The first time this filing system is used with a disc track it must be initialised to the format required. To do this, type :-

DISCININ([<n1><n2>...]); where<n1>,<n2>, ... are the numbers of the tracks to be initialised.

The system is now ready to accept and handle files in this and subsequent POP-2 sessions. The first n sectors of each track so initialised are used by the system for storing the track's file directory. n is held in the standard variable DISCSTART and is set initially to be 10. The rest of the track will be used for your files. You may at any time add further tracks to your filing system by calling DISCINIT again.

Changing tracks

There is a separate directory for each track in use. You may change from one track to another by typing :-

DTRACK(<n>); where <n> is the number of the track you wish to use. (See example 3).

The standard variable DISCUSER holds the number of the current track.

The standard variable DTRS holds either an integer or a list of integers. If an attempt is made to input a file which is not on the current track, all the tracks in DTRS will be examined before an error is given.

Commands available

In this description <filename> denotes a file name (e.g. RMB THEOREMS or APAP SYS 4).

A. Commands which create new disc files

DREAD([<filename>]); reads characters from the console until a halt-code is depressed (CTRL and T). The characters are put into a file (on disc) called <filename>. If a file with this name already exists on any of the tracks in DTRS it is lost when the halt-code is read from the console.

DPTIN([<filename>]); inputs the file <filename> from paper tape and creates a disc file with the same name.

DJOIN

DJOIN([<file1>],[<file2>],[<file3>]); creates a new disc file which contains the characters in <file1> followed by the characters in <file2> and calls it <file3>. The two originating files are not affected unless <file3> is the same name as one of the others, when the original file is deleted.

DREPIN([<filename>],<charrep>); creates a disc file with the name <filename> from the character repeater <charrep>. This command can be used to put files already on disc into the filing system.

DCOPY([<filename1>],[<filename2>]); copies the file <filename1> to the file <filename2>.

DEDIT([<filename]); is a command which creates a new disc file. Its use is described below under D.

When the transfers involved in the above commands have been completed (i.e. when a halt-code or TERMIN has been read) the message

[<filename>]<discend>

is output on the console. For the significance of DISCEND see below under HOUSEKEEPING. If a transfer to the disc is interrupted before it is terminated (e.g. by pressing CTRL and G) then the new file is not entered in the directory.

B. Commands to copy a disc file onto another device

DTYPE([<filename>]); outputs to the console the file <filename>. To interrupt just depress CTRL and G.

DLP80([<filename>]); lists <filename> on the lineprinter.

DPTOUT([<filename>]); outputs <filename> onto paper tape.

C. Commands for compiling disc files

DCOMP([<filename>]); compiles the file <filename>. When the compilation has been completed the message :-

[<filename>]COMPILED

is output on the console.

D. Commands for editing disc files

DEDIT([<filename>]); edits a file discarding the previous version. It expects edit commands from the keyboard in the format of POPEDIT (see File editing facilities). When the editing has been completed, the message :-

[<filename>]<discend>

is output on the console. For the significance of DISCEND see below under HOUSEKEEPING. The original file is not lost until this message

has been output and it is therefore possible to recover from an obviously wrong edit merely by interrupting it - depressing CTRL and G. (To recover after the message has been output DRECOVER will have to be used - see below).

DRECOVER([<filename>]); puts back into the directory the version of <filename> which has just been displaced by editing. The edited version is discarded. DRECOVER can only be used successfully if there has been no intervening call of DEDIT, DTIDY (see below) or DTRACK. If there has been such a call, or if the file is wrongly named, then the message :-

SORRY CANT RECOVER [<FILENAME>]

is output.

E. Other commands to handle files

DIN([<filename>]); has as result the input character repeater of the file <filename>. DIN is analogous to DISCIN.

DOUT([<filename>]); has as result the output character repeater of the file <filename>. DOUT is analogous to DISCOUT.

DKILL([<filename>]); discards the file <filename>. The area of disc on which it was stored will become available for other files.

Housekeeping

Files are added to the disc track consecutively, and the sectors freed by editing or killing files are not automatically made available for re-use. To do this you must give the command :-

.DTIDY;

which shuffles the files and makes the freed space available. (See example 6). It is desirable to use this command whenever your track is in danger of becoming full. The variable DISCEND points to the next free sector on the track being used. Whenever a new file is added to the track the new value of DISCEND is printed so that you may know how full your track is. DTIDY also prints the new value of DISCEND. (There are 160 sectors on a disc track).

The system uses a directory which stores the actual position and size (in sectors) of each file and free area.

DISCDIR =>

prints out this file directory. (See example 6).

A standard variable DDIRSTART, set to zero by the system, contains the sector number in which the directory starts. This can be altered by the user, but files from other users' tracks cannot then be accessed. DDF1 and DDF2 contain the input and output streams respectively. A full description is given in the library documentation of EASYFILE.

Errors

An attempt to call a file by a name which is not a list causes ERROR 54. An attempt to access a file which is not in the directory produces ERROR 57. Overfilling the track causes ERROR 79.

Suggestions for efficient use of the system

It should be more convenient to keep a large program as a lot of short files with the master file to compile them (or list them). (See example 5). Not only is editing much easier, but also the progress of compilation of a large program is indicated by the messages output as each subsidiary file is compiled, also, if a large file is more than 80 sectors long, it will not be possible to edit it, as two copies of the file cannot exist on the 160 sectors of the track.

Examples

In the examples which follow, a halt-code (depressing CTRL and T) is indicated by Ⓗ. Characters typed in by the user are underlined, to distinguish them from output from the machine.

Example 1 Initialising a track, reading a file in from the console, editing and compiling it.

```
: DISCINIT([94]);
: DREAD([APA 1]);
: THE CAT SAT ON THE MAT].HD.PR;
:Ⓗ
[APA 1] 11
: DTYPE([APA 1]);
THE CAT SAT ON THE MAT].HD.PR;
: DEDIT([APA 1]);
EDIT:-
```

```
: IS[
: SH
[APA 1] 12
: DCOMP([APA 1]);
THE
[APA 1] COMPILED:
```

Example 2 A subsequent POP-2 session. Reading in a file from disc. Use of DRECOVER.

```
: DTRACK(94);
: DREPIN([SYSTEM], DISCIN(94,100));   (Bring file SYSTEM under
[[SYSTEM] DISCEND 38]                  the control of the filing
: DEDIT([SYSTEM]);                     system from track 94
EDIT:-                                 sector 100 on the disc.)
: FL THEN                              (Edit this file.)
: DC X
: SH
EDITED
[SYSTEM] 45
: DCOMP([SYSTEM]);                     (Attempt to compile the
ERROR 22                               new file SYSTEM uncovers
IN FUNCTION SYS                        a syntax error in it.)
CULPRIT CLOSE
SETPOP:
: DRECOVER([SYSTEM]);                  (Recover the old file SYSTEM.)
: DCOMP([SYSTEM]);
[SYSTEM] COMPILED:                     (This file compiles correctly.)
```

Example 3 Changing tracks (supposing that 93 and 94 have been initialized at some earlier POP-2 session).

```
: DTRACK(94);
: DPTIN([APA 2]);
[APA 2] 141
```

```
: DTRACK(93);                    (Change the track on which
: DPTIN([THEOREMS]);              the system is currently
[THEOREMS] 33                     operating from 94 to 93,
:                                 and read in a paper-tape
                                  file to that track).
```

Example 4 Use of DTIDY.

```
:DPTIN([SYSTEM 2]);
[SYSTEM 2] 152
: .DTIDY;
DISCEND 127
:
```

Example 5 Storing a large program as several small files. Assume that there are on the disc the files [APA SYS1], ... [APA SYS6] and they together comprise a program [APA SYSTEM]. Make up a master file [APA SYSTEM] as follows :-

```
:DREAD([APA SYSTEM]);
: GENFUN([APA SYS1]);  GENFUN([APA SYS2]);      (GENFUN is an arbitrary
: GENFUN([APA SYS3]);  GENFUN([APA SYS4]);       name for a function
: GENFUN([APA SYS5]);  GENFUN([APA SYS6]); Ⓗ     which can be redefined
[APA SYSTEM] 97                                  so that the file can
:                                                perform various
                                                 actions.)
```

If you wish to compile [APA SYSTEM]

```
:DCOMP -> GENFUN;                (Define GENFUN to be
: DCOMP([APA SYSTEM]);            DCOMP. This will
[APA SYS1] COMPILED               cause the various
[APA SYS2] COMPILED               sub-files to be com-
  :                               piled when SYS is
[APA SYS6] COMPILED               compiled.)
[APA SYSTEM] COMPILED
:
```

If you wish to list [APA SYSTEM] on the lineprinter do :-

```
: DLP80 -> GENFUN; DCOMP([APA SYSTEM]);
[APA SYSTEM] COMPILED
:
```

(GENFUN is defined to be DLP80, this will cause a listing of the sub-files.)

If you wish to copy [APA SYSTEM] to paper tape do :-

```
: DPTOUT -> GENFUN;  DCOMP([APA SYSTEM]);

etc.
```

If you wish to edit [APA SYSTEM] you need only edit the relevant subsidiary file - say [APA SYS4]. Since the new version of [APA SYS4] has the same name, you do not need to alter your master file.

Example 6 The disc file directory and DTIDY. This example shows how the directory develops, starting from a newly initialized track.

```
: DISCINIT([94]);
: DISCDIR =>
** NIL,
: DPTIN([APA 1]);
[APA 1] 15
: DPTIN([APA 2]);
[APA 2] 27
: DISCDIR =>
**  [[[APA 2] 15 12][[APA A] 10 5]],
: DPTIN([APA 1]);
[APA 1] 32
: DISCDIR =>
**  [[[APA 1] 27 5][[APA 2] 15 12][FREE 10 5]],
: .DTIDY;
DISCEND 27
: DISCDIR =>
**  [[[APA 1] 22 5][[APA 2] 10 12]],
:
```

POP-2 PROGRAM LIBRARY

This is a collection of both utility and research programs, the first word of whose name is always LIB, e.g. [LIB SETS]. They are available both in paper tape form and are also stored on the disc, for which a standard function LIBRARY is provided to access them; LIBRARY takes a program name as its argument and produces the corresponding character repeater as its result.

```
e.g.  LIBRARY([LIB SETS]) -> IN;
or    COMPILE(LIBRARY([LIB EASYFILE]));
```

For convenience the word LIB may be omitted when using the function LIBRARY.

If the paper-tape copy is required, POPMESS should be used in the standard way, the word LIB must be included.

```
e.g.  POPMESS([PTIN LIB SETS]) -> IN;
or    COMPILE(POPMESS([PTIN LIB EASYFILE]));
```

A list of programs available, and their documentation, can be found in the folder 'POP-2 PROGRAM LIBRARY' which accompanies all Multi-POP consoles.

Alternatively, a list of the current programs and a short synopsis for each may be obtained by compiling the library program [LIB CONTENTS].

```
e.g.  COMPILE(LIBRARY([CONTENTS]));
```

FILE EDITING FACILITIES

A set of standard functions are provided for sequential editing of POP-2 files. The file to be edited, and the editing instructions may come from any input device, and the edited file may be output to any number of devices.

Two functions, POPGOBBLE and POPEDIT, are provided. These give all the required facilities.

POPGOBBLE takes an input character repeater as its argument, and applies this until the end of the associated file is reached. It is used to 'drive' the editor when no compilation is required; this is made clear by examples later.

POPEDIT takes three arguments. These are :-

1) The character repeater of the file to be edited.
2) The character repeater of the edit commands.
3) An output character repeater, or a list of such repeaters for the devices to which the edited file is to be output (this may be NIL).

The result of POPEDIT is the character repeater of the edited file, and when it is used (e.g. by COMPILE or POPGOBBLE), it produces the characters of the original file modified by the edit commands, copying the characters to the output devices as a side effect.

Examples of the use of POPEDIT and POPGOBBLE are given later.

If the edit commands are being typed in from the console, (i.e. the character repeater for the edit commands is CHARIN), the program outputs the following message after the character repeater supplied by POPEDIT has been called :-

EDIT:-

The first edit command should now be typed in, followed by carriage-return. When this command has been obeyed, ":" is output and the next command should be given, etc.

If the edit commands are being given off-line, (i.e. from a paper-tape or disc file), they are read automatically when required.

When the file being edited has successfully been read to its end, all the input and output files are closed, and the following message is output to the console :-

EDITED

Edit commands

The type of edit commands available are the same as those in the Elliott program "EDIT41".

An edit command consists of three parts: a function part, a space (which may be omitted), and a string of characters. Two characters specify the function, and after the space, the remaining characters up to, but not including the next new line character, form the string.

The edit "functions" provided are :-

FL DL FC DC FE DE IS IL IB SH

These have the following effects :-

FL - Find line: The input file is copied until a line beginning with the edit string is found. The last character copied is the last character of the edit string.

DL - Delete to line: The input file is skipped until a line beginning with the edit string is found. The last character skipped is the last character of the edit string.

FC - Find characters successively: If the characters of the edit string are C_1, $C_2 \ldots C_n$, then the input file is copied until C_1 has been copied, and then further until C_2 has been copied and so on until C_n has been copied.

If no edit string is given, one character is copied.

DC - Delete to characters successively: If the characters of the edit string are C_1, $C_2 \ldots C_n$, then the input file is skipped until C_1 has been skipped, and then further until C_2 has been skipped, and so on until C_n has been skipped. If no edit string is given, one character is deleted.

FE - Find end of line: The input file is copied up to, but not including the next new line character. The new line character is read and stored, and will always be regarded as the next character from the input device.

DE - Delete to end of line: The input file is skipped up to but not including the next new line character. The treatment of this new line character is the same as for FE.

IS - Insert on same line: The edit string is copied.

IL - Insert on a new line: A new line character is output and then the edit string is copied.

IB - Insert a block: The edit string, including new line characters is copied. The string must be terminated by a combination of a new line followed by a "↑" character. The new line and "↑" characters are not output.

SH - Reads the remainder of the input file up to and including its terminating character.

Ignorable characters

Spaces are copied, but are ignored for search purposes. For example, when searching for a line which is indented, it is not necessary

to put the preliminary spaces into the edit string, but if spaces are given at the beginning of a line which is to be inserted, they will be output; remember that if 4 spaces are required, 5 should be given after the edit function as a space is expected to separate it from the string.

Additional commands - ON and OFF

The commands ON and OFF can be inserted in the edit commands. They do not have any edit string and are used to monitor the editing process.

ON - causes the edited stream to be copied onto the console in addition to any other devices which are being used to copy the output.

OFF - causes the copying onto the console to stop.

Errors during editing

If an unacceptable edit command is read, then the following message is output :-

POPEDERR CULPRIT <x> where <x> is the offending character.

If the edit stream is being input from the teletype, then

TRY AGAIN

is output and the user should retype the edit command. If, however, the edit stream was coming from another device, the message :-

CONTINUE BY TYPING EDIT COMMANDS

is output, the edit file closed, and the edit file reverts to the teletype. The user must continue his edit by typing in the commands from the teletype.

If the end of the source file is read while any command other than SH is being obeyed, then the message :-

END OF SOURCE FILE. OUTPUT FILES CLOSED

is output, and the edit is terminated. The user must take any necessary action to recover the original files. This error is most likely to be caused by a FL command, the string given not being correct.

If the end of the edit commands file is reached before the end of the source file is reached, the message :-

```
END OF EDIT FILE. CONTINUE BY TYPING COMMANDS
```

is output. The user must complete the edit by typing commands from the teletype.

Double editing

It is not possible to do double editing using POPEDIT.

Examples of use of file editing facilities

In the following examples, it is assumed that the file to be edited (the source file) is represented by the character repeater INFILE, various output files are represented by the character repeaters OUTLP, OUTPT, OUTDISC etc., and the file of edit commands is either the character repeater CHARIN, if the editing is being done on-line, or EDITCOMMANDS.

These character repeaters are all created in the standard way :-

e.g. for paper-tape

```
POPMESS([PTIN INPUTFILE]) -> INFILE;
POPMESS([PTIN EDITS]) -> EDITCOMMANDS;
POPMESS([PTOUT EDITED FILE]) -> OUTPT;
```

e.g. for disc

```
DISCIN(<t>,<n>) -> INFILE;
DISCOUT(<t>,<n>)->OUTDISC;
```

where <t> and <n> are the required disc track and sector numbers.

e.g. for lineprinter

```
POPMESS([LP80 EDITED FILE]) -> OUTLP;
```

A) On-line editing

1) To edit a file using the teletype, and to compile the edited file without outputting it to any device, type :-

```
COMPILE(POPEDIT(INFILE,CHARIN,NIL));
```

This should be followed by the edit commands.

2) To edit a file using the teletype, to compile the edited file and to copy the new file to the disc, type :-

```
COMPILE(POPEDIT(INFILE,CHARIN,OUTDISC));
```

This should again be followed by the edit commands.

3) To edit a file from the teletype, to compile the edited file, and to copy the new file onto both the disc and the lineprinter, type :-

```
COMPILE(POPEDIT(INFILE,CHARIN,[%OUTDISC,OUTLP%]));
```

Again, follow this with the edit commands.

4) If simultaneous compilation, as above, is not required, COMPILE, in all cases, should be replaced by POPGOBBLE.

For example, to edit a file without compiling it, but producing a new file on disc, a paper-tape copy, and a listing to the lineprinter, the user would type :-

```
POPGOBBLE(POPEDIT(INFILE,CHARIN,[%DISCOUT,OUTLP,OUTPT%]));
```

followed by the edit commands.

B) Off-line editing

If the user wishes to use an off-line file of edit commands, e.g. a paper-tape which he had typed up previously, in all cases in the above examples, the character repeater for the teletype, CHARIN, would be replaced by that for the file of edit commands :-

```
e.g.  COMPILE(POPEDIT(INFILE,EDITCOMMANDS,OUTLP));
```

If the edit is successful, then the message :-

```
EDITED
```

will be output, otherwise an error message as described, will be given.

Complete examples of the use of POPEDIT

The following file is assumed to be on track TR sector N of the disc :-

```
FUNCTION SIGMA L;
  IF L.NULL THEN 0 ELSE L.HD + SIGMA(TL(L))
  CLOSE
END;

FUNCTION FACT N; N*FACT(N-1);
END;
"A" -> A;
1,2,3 -> L;
FACT(A) =>
SIGMA(LIST) =>
```

It is required to edit this file, compiling it, and copying it back to disc track TR1, sector N1, producing a paper-tape copy.

The required input on the teletype is given below, assuming editing is to be done online, and for clarity, the user's output is underlined :

```
: VARS AA BB CC;
: DISCIN(TR,N) -> AA;
: DISCOUT(TR1,N1) -> BB;
: POPMESS([PTOUT EDITED FILE]) -> CC;     (Giving the paper tape
                                           file the name [EDITED FILE]
: COMPILE(POPEDIT(AA,CHARIN,[%BB,CC%])); (Calling POPEDIT

EDIT:-                                    (Output by POPEDIT to
                                           inform user that he may
                                           start typing in commands
: FL END
: FE
: ON
: FC;

FUNCTION FACT N; : IL IF N=0 THEN 1 ELSE (Output due to the ON
                                          command followed by the
IF N=0 THEN 1 ELSE : FC )                 next edit commands
N*FACT(N-1) : IS CLOSE                    after the ":"
CLOSE : OFF
: FC E;                                   (Switch off teletype
: IB                                       output

VARS A LIST;
  2 -> A; [%1,2↑2,3%] -> LIST;

↑                                         (Inserted block terminated
: DL 1                                     with ↑
: DE
: SH
```

```
** 2,                          (Output due to the two print
                                statements in the source file
** 8.0,

EDITED
:
```

The new file produced by the above edit commands on the given file is given below. This file will have been punched to paper-tape, and also written onto the disc :-

```
FUNCTION SIGMA L;
  IF L.NULL THEN 0 ELSE L.HD + SIGMA(TL(L))
  CLOSE
END;

FUNCTION FACT N;
  IF N=0 THEN 1 ELSE N*FACT(N-1) CLOSE;
END;

VARS A LIST;
  2 -> A; [%1,2↑2,3%] -> LIST;

FACT(A) =>
SIGMA(LIST) =>
```

DEBUGGING FACILITIES

The DEBUG system is a set of standard macros, functions, and variables, which allow the user to trace specified functions by outputting the values of selected parameters and results, as the functions are obeyed.

How to use the debugging facilities

The user first specifies any functions he may wish to have traced, by using the macro SPEC.

For example, consider the function :-

```
FUNCTION ADD X Y;
  X+Y
END;
```

Using the output local list facility in the language, this can be written as :-

```
FUNCTION ADD X Y => RESULT
  X+Y -> RESULT;
END;
```

To specify ADD one would do :-

```
SPEC ADD X Y => RESULT;
```

NOTE: The function specified does not need to be written in the second form (using the output local list facility), in order that it may be specified.

Similarly, if the function F has four arguments, say A, B, C and D, and produces three results, say X, Y and Z, then F may be specified by doing :-

```
SPEC F A B C D => X Y Z;
```

It is not necessary to specify all parameters and results, just the last m parameters, and n results may be specified if desired ($m \geq 0, n \geq 0$). Also, if in the specification, a parameter or result is given as the item *, then that parameter or result will not be printed when tracing.

For example, if with function F above, it is only required to trace the values of C, X and Z, then the specification would appear as :-

```
SPEC F C * => X * Z;
```

Specifying a function has no effect on its running speed, and only uses a few words of store for each function, so that it can well be done after each function, or group of functions, when writing the program in the first place. The FNPROPS (see page 22) of the function is used to hold the information.

To cause a particular function, or functions, to be traced, the macro BUG should be used in the following way :-

```
BUG ADD;          will cause the function ADD to be traced,

BUG ADD F1 F;     will cause the functions ADD, F1, and F,
                  to be traced.

etc.
```

From then on, tracing will occur on entry to, and on exit from, the bugged functions provided that the variable DEBUG is set to TRUE.

Thus the tracing can be controlled over all functions bugged, by setting the variable DEBUG to FALSE, for no output, or TRUE, for output.

The printout given while tracing is, on entry to the function, the name of the function on a new line, preceded by the symbol '>', and followed by the names of the selected arguments and their values. On exit from the function the name of the function is printed on a new line preceded by the symbol '<', and is followed by the names of the selected results and their values.

If a function which has not been specified with SPEC is bugged, then the printout consists of the name of the function only.

On each function entry the printout is indented by one space, and on exit the indentation is reduced by one space, the amount of indentation being held in the variable DEBSP. The user may change the value of this variable during a trace if a special lay-out is required. DEBSP is automatically set to zero if a call of SETPOP occurs, but may be assigned values by the user.

The values of parameters and results are printed during tracing by the function DEBPR, which initially has the value PR. This function may be redefined by the user if it is required to print special values, arrays, or records for example. The definition given to DEBPR must be a function which takes one argument and leaves no results. If this is not done the result of the debugging will be undefined.

To stop tracing a particular function, or functions, the macro UNBUG should be used in a similar way to BUG, e.g.

```
UNBUG ADD;        will cause the debugging mechanism to be
                  removed from ADD,

UNBUG FF GETY;    removes the debugging mechanism from the
                  functions FF and GETY.

etc.
```

It should be noted that BUG and UNBUG affect the function stored in the function variable, and if the value of the function has to be taken out and used elsewhere, e.g. by partially applying it, or tying it up in a data structure, they will not affect the incorporated function.

POPTRACE (see page 24) can be used to list the calling sequence of currently active functions, and if POPDOTRACE is set to TRUE this will be entered by the system when an error occurs.

Examples of the use of the debugging facilities

```
: FUNCTION ADD X Y;
:  X+Y
: END;
:
: SPEC ADD X Y => SUM;                (Specify the function ADD)
:
: FUNCTION ADD3 U V W;
:  ADD(U,ADD(V,W))
: END;
:
: SPEC ADD3 V * => SUM3;              (Specify the function ADD3.
                                       V is the only argument whose
                                       value is required)
: BUG ADD ADD3;                       (Set the debugging mechanism
:                                      on ADD and ADD3)
: TRUE -> DEBUG;                      (Switch on the debugging
:                                      mechanism)
: ADD3(1,2,3)=>

  >ADD3 V= 2, *,
   >ADD X= 2, Y= 3,
   <ADD SUM= 5,
   >ADD X= 1, Y= 5,
   <ADD SUM= 6,
  >ADD3 SUM3= 6,

** 6,

:  UNBUG ADD;                         (Reset the function ADD)
: ADD3(2,3,4) =>

  >ADD3 V= 3, *,
  <ADD3 SUM3= 9,

** 9,
:
```

TIMING FACILITIES

The amount of processor time used during a session is held in the standard variable POPTIME as an integral number of time units. Each unit represents one-sixteenth of a second, POPTIME being incremented every 'tick' of the real-time clock.

Any process may be timed by comparing the values of POPTIME before and after the process is run, but because of the rather large size of the unit, POPTIME should always only be regarded as an approximate measure of the time taken, particularly for short processes.

A further facility, POPDATE, is provided. This is a function which produces as its result a list of the real-time, the day, the month and the year.

```
e.g.    : POPDATE() =>

        ** [12.35 25 JAN 1970],
        :
```

A standard variable LAPSETIME holds a count of real-time since the system came on the air. It is incremented every 16th of a second.

```
e.g.    : POPTIME,LAPSETIME =>

        ** 23,1716,
        : POPTIME,LAPSETIME =>

        ** 24,1793,
        :
```

PROGRAM CORE REQUIREMENTS

Listed below is the amount of core required by standard data structures. These figures can be used to determine roughly how much core a program and its workspace will occupy, but it should be noted that during the construction of many of these structures, functions in particular, much more core will temporarily be required.

STRUCTURE	CORE REQUIREMENT IN WORDS	COMMENTS
ARRAYS	$36+3d+e_1(2+e_2(2+\ldots e_{d-1}(2+e_d))\ldots)$	where d = No. of dimensions and e_1 = No. of elements in first dimension, e_2 = No. of elements in second dimension, etc.
DOUBLETS	When two functions are combined to produce a doublet, no extra core is required.	
CLOSURE FUNCTIONS	$6+2n$	where n = No. of frozen formals. For doublets this figure should be doubled.
FUNCTIONS	A good approximation to the amount of core required for a function can be obtained by counting the number of	

STRUCTURE	CORE REQUIREMENT IN WORDS	COMMENTS
	words in the function definition including everything other than labels, brackets, commas, semi-colons and the word CLOSE; count in the number of formal parameters twice and add six.	
LISTS	3 words/cell	i.e. for each PAIR
REFERENCES	2	
RECORDS	23+20c	when the record class if created, and for each
	1+c	record created where C = No. of components in the record class.
STRIPS	25	when creating a new strip class, and
	2+n	for each strip of n full size components, and
	3+INTOF(n*s/24+0.5)	for each strip of n components of size s bits.
WORDS	6	when a word is used as a variable name, another 2 locations are required. When the MEANING of a word is first updated, another 3 locations are required. The first time that MEANING is used for any word an extra 65 locations are required.

ADDITIONAL DEFICIENCIES AND CHANGES IN DEFINITION

In addition to those specified in their relevant sections in this manual, the following deficiencies and changes in definition exist in this implementation of POP-2 :-

1) SECTIONs have been re-defined (see page 23).

2) Machine code is not allowed.

3) POPVAL must be given text which is a completed imperative sequence, otherwise the result is undefined.

4) The components of records always occupy one full location irrespective of their size as given in RECORDFNS.

5) Concatenation of two lists using the infix <> copies the list which is its first argument, e.g. in performing A<>B, the list A is copied. (This produces a useful method for copying a full list - e.g. A<>NIL). It should be noted that methods for appending items to a list which take the form L<>[%A%] are thus very inefficient.

6) Strips may be created with components whose size lies within the range 0 (full items) to 22 only.

ADDITIONAL AVAILABLE STANDARD FUNCTIONS

All the optional functions mentioned in Appendix 2 of the Reference Manual are provided, except arctan (which is in library program [LIB INVTRIG]).

EQ(X,Y); equivalent to NONOP=(X,Y);

ISNUMBER(X); equivalent to ISINTEGER(X) OR ISREAL(X)

OCPR(X); prints the item X to the current output device as an 8 digit octal number

POPNEWS; A standard MACRO which causes a current Multi-POP news message to be printed on the console.

SWAPOFF(); a function which when applied, causes the user's current time-slot in the scheduling of Multi-POP to be terminated (mainly for use in systems type programs).

POPUSER; A standard variable which contains the users logging-on identifier.

POPEXECUTE; A standard variable which is false during compilation and true otherwise.

5) Concatenation of two lists using the infix <> copies the list which is its first argument, e.g. in performing A<>B, the list A is copied. (This produces a useful method for copying a full list - e.g. A<>NIL). It should be noted that methods for appending items to a list which take the form L<>[%A%] are thus very inefficient.

6) Strips may be created with components whose size lies within the range 0 (full items) to 22 only.

ADDITIONAL AVAILABLE STANDARD FUNCTIONS

All the optional functions mentioned in Appendix 2 of the Reference Manual are provided, except arctan (which is in library program [LIB INVTRIG]).

EQ(X,Y); equivalent to NONOP=(X,Y);

ISNUMBER(X); equivalent to ISINTEGER(X) OR ISREAL(X)

OCPR(X); prints the item X to the current output device as an 8 digit octal number

POPNEWS; A standard MACRO which causes a current Multi-POP news message to be printed on the console.

SWAPOFF(); a function which when applied, causes the user's current time-slot in the scheduling of Multi-POP to be terminated (mainly for use in systems type programs).

POPUSER; A standard variable which contains the users logging-on identifier.

POPEXECUTE; A standard variable which is false during compilation and true otherwise.

REFERENCES

Burstall, R.M., Collins, J.S. and Popplestone, R.J. (1968) POP-2 Papers. Edinburgh: Edinburgh University Press.

Burstall, R.M., and Popplestone, R.J. (1971) POP-2 Reference Manual. Programming in POP-2 (ed. R.M. Burstall) Edinburgh: Edinburgh University Press.

Burstall, R.M. and Collins, J.S. (1971) A primer of POP-2 Programming, Programming in POP-2 (ed. R.M. Burstall) Edinburgh: Edinburgh University Press.

OTHER DOCUMENTATION

Anderson, B. (1970) Programming in POP-2, Computer Weekly. (A series in Computer Weekly based on Programming in POP-2).

Barnes, J.G.P., Steel, R. (1968) System 4 POP-2 Users Guide. Edinburgh: School of Artificial Intelligence.

Marsh, D.L. (1970) An introduction to POP-2: Computer Weekly, 172, 6 & 7.

"POP-2" Information booklet published by Conversational Software Ltd., 12 Queen Street, Edinburgh, EH2 1JE. (1970).

Popplestone, R.J. (1968) The design philosophy of POP-2. Machine Intelligence 3 (ed. D. Michie) Edinburgh: Edinburgh University Press, pp. 393-402.

Pullin, D.J.S. (1967) A plain man's guide to Multi-POP implementation. Mini-MAC Report No. 2. Edinburgh: School of Artificial Intelligence.

Scott, J.J. (1968) A supplementary manual for the Lancaster POP-2 1900 System. University of Lancaster.

APPENDIX 1. A COMPLETE MULTI-POP SESSION

```
↑G                                  (User hits bell to activate console).

MULTI-POP SYSTEM. ISSUE 21/11/69.  16.46HRS.  25 JAN 1970

NAME:  RDD                          (System requests name and
                                     amount of store required)
STORE 30 BLOCKS FREE:  3

DISC TRACK:  0

SETPOP:
:
: 12+2.5*(1.5+2.5) =>               (Simple arithmetic)

**  22.0,
:
: VARS A B SUM;                     (Declare and give values to
: 2*2 -> A; 3*A -> B; A*A+B*B -> SUM;  some variables)
: SUM =>

**  160,
: FUNCTION SUMSQ X Y;               (Define function for computing
:  X*X+Y*Y                          the sum of the squares of two
: END;                              numbers, and test it)
: SUMSQ(A,B) =>

**  160,
:
: FUNCTION FACT N;                  (Define factorial function)
:  IF FAC!
:  IF N=0 THEM←N 1 ELSE N*FACT(N-1) CLOSE
: END;
: FACT(FACT(3)) =>                  (Test factorial function)

**  720,
:
: 1 -> I;                           (Assign value to variable I)

COMMENTS [I]                        (System commenting on use of
                                     undeclared variable)
: VARS U;
: [%I,I+1,"DOG","CAR←T"%] -> U;     (Create a list, assign to U)
: U =>

**  [1 2 DOG CAT],
:
: [1 2 3 4]<>U =>                   (Concatenate two lists)

**  [1 2 3 4 1 2 DOG CAT],
```

APPENDIX 1

```
: FUNCTION TAB F LO STEP HI;                    (Declare function for tabulat-
:  LOOP:                                         ing any uni-argument, uni-
:   1.NL; PR(LO); PR(F(LO)); LO+STEP->LO;        result function over a given
:   IF LO=<HI THEN GOTO LOOP CLOSE               range)
: END;
:
: TAB(SQRT,1,100);

ERROR 52                                        (Too few arguments provided)
CULPRIT FNENTRY
SETPOP:
:
: TAB(SQRT,1,1,100);

 1 1.0
 2 1.414
 3 1.732
 4 2.0
 5 2.236                                        (Caused Interrupt by pressing
 6 2.449                                         CTRL G twice.  Output stops
 7 2.646                                         at once)
 8 2.828
 9 3.0
10 3.162
11 3.317
12 3.464
13 3.606
14 3.74↑G
SETPOP:
: POPMESS([LP80 SQUARE ROOTS]) -> LP;           (Open lineprinter file)

COMMENTS [LP]
: LP -> CUCHAROUT;
:
: TAB(SQRT,1,1,100);                            (Output results to lineprinter)
:
: TERMIN.LP;                                    (Close lineprinter file and
: CHAROUT -> CUCHAROUT;                         reset CUCHAROUT to the console)
:
: 2+2 =>

**  4,
: .LOGOFF;                                      (Log off the system)

CPU TIME USED = 0 MINS 3.313 SECS.

LOGOFF 16.58HRS. 25JAN 1970

:
```

STANDARD POP-2 CHARACTER SET

Character	Internal Value			Character	Internal Value	
	Decimal	Octal			Decimal	Octal
0	0	00		`	32	40
1	1	01		A	33	41
2	2	02		B	34	42
3	3	03		C	35	43
4	4	04		D	36	44
5	5	05		E	37	45
6	6	06		F	38	46
7	7	07		G	39	47
8	8	10		H	40	50
9	9	11		I	41	51
:	10	12		J	42	52
;	11	13		K	43	53
<	12	14		L	44	54
=	13	15		M	45	55
>	14	16		N	46	56
10	15	17		O	47	57
Space	16	20		P	48	60
Newline	17	21		Q	49	61
"	18	22		R	50	62
RESERVED	19	23		S	51	63
£	20	24		T	52	64
%	21	25		U	53	65
&	22	26		V	54	66
'	23	27		W	55	67
(	24	30		X	56	70
)	25	31		Y	57	71
*	26	32		Z	58	72
+	27	33		[	59	73
,	28	34		$	60	74
-	29	35		]	61	75
.	30	36		↑	62	76
/	31	37		Shift	63	77

NOTES

The reserved character (decimal 19) is used by the system for file termination, and will produce an undefined result if output directly by the user.

The shift character (decimal 63) may be used by the user as a special purpose marker, as no out-shift facilities are envisaged. If printed this character will be output as a space.

If decimal 64 is output to the lineprinter, a page throw will be given, on all other devices it will be interpreted as the digit 0, if a negative number is output as a character to the lineprinter, this has the effect of carriage return without line feed and can thus be used for overprinting.

POP-2 ERROR NUMBERS

1 Impermissible use of :
2 Illegal position for a label
3 More than one label with the same name
4 Type clash in variable declaration
5 Impermissible use of ->
6 Impermissible operation in execute mode
7 MACRO definition not at execute level, or attempt to specify formal parameters in a MACRO definition.
8 Impermissible use of =>
9 Missing "
10 OPERATION not followed by precedence, or followed by impermissible precedence
11 NONOP not followed by an operation, or . not followed by an identifier
12 Impermissible use of CANCEL
13 GOTO with undefined label
14 Attempt to SWITCH on impermissible item, i.e. non-integral, non-positive or too large
15 Illegal mnemonic (M/C code)
16 Illegal variant (M/C code)
17 Illegal address for short instruction (M/C code)
18 Illegal offset (M/C code)
19 Missing semi-colon (M/C code)
20 Missing separator
21 Impermissible separator or expression terminator
22 Impermissible closing bracket
i.e. missing END, THEN, CLOSE,],), %], %) or ENDSECTION
23 Attempt to JUMPOUT or REINSTATE over a SECTION boundary
24 GOON at execute level with the stack empty
25 Impermissible use of systems name
26 Type clash (Loader)
27 Impermissible control character (Loader)
28 Non-function argument for FNTOLIST
29 " " " " update part of UPDATER
30 " " " " UPDATER
31 Attempt to assign non-function to an OPERATION or MACRO-type variable, or to an updatable system function
32 Non-function argument for INCHARITEM or OUTCHARITEM
33 Arithmetic overflow
34 Integer overflow (INTOF)
35 Last argument for CONSWORD or CHARWORD is not a legal positive integer
36 Attempt to look up item other than in dictionary (System error)
37 Attempt to obey undefined function, or non-function
38 Assignment with non doublet, i.e. attempt to obey undefined update part of function

39 Impermissible argument - SQRT
40 " " - SIN
41 " " - COS
42 " " - TAN
43 " " - EXP
44 " " - LOG
45 " " - general selector or updating function
e.g. HD , TL, etc.
46 Impermissible argument - DESTWORD
47 Non-number argument for arithmetic operation
48 Attempt to partially apply a non-function
49 Non-integer argument for // or second argument of PRBIN
or PROCT is not a legal positive integer
50 No room in store for required operation
(Can be caused by any operation requiring store. Continuation
is impossible until required amount of store becomes free.
The culprit given is the number of words of store which was
being looked for)
51 Stack overflow
52 Stack underflow
53 Impermissible component size (<0 or >22), for record class
- RECORDFNS
54 Item read by LISTREAD is not a list, or
item read by NUMBERREAD is not a number
55 Attempt to use a file which has been closed
56 Peripneral device in error state, or parity error
57 Wrong file loaded by operator, or non-existent EASYFILE file
58 POPMESS of wrong format
59 Device requested by POPMESS not available
60 Impermissible argument - FNPROPS
61 JUMPOUT fail
62 Impermissible argument - structure processing function
e.g. DATALIST, COPY
63 Impermissible component size - Strip update function
64 Impermissible argument - Strip initiator function
65 Impermissible component size - STRIPFNS
66 Impermissible strip class - Strip selector function
67 " " " - Strip update function
68 " subscript - Strip selector function
69 " " - Strip update function
70 " use of %, ,, or string quotes
71 Ambiguous use of the item .
72 Error in 2: or 8: integers
73 Impermissible subscript - array access
74 " " - array update
75 " boundslist - array declaration
76 FROZVAL fail (Impermissible subscript, or attempt
to apply to non-closure function)
77 Attempt to update FNPART with non-function
78 Attempt to apply FNPART to non-closure function

79	Attempt to read or write beyond the end of a disc track
80	Attempt to write onto a disc track which is not assigned to you
81	Disc transfer failed (hardware error)
82	Attempted disc transfer using non-existent track or sector
83	
84	Attempt to transfer an illegal structure between disc and core
85	Impermissible sector offset given for transfer between disc and structure
86	Corruption detected - DOTRACE
150	Attempt to apply APPSTATE or REINSTATE outside their proper barriers
151	Attempt to REINSTATE an item that is not a state

APPENDIX 4. POP-2/4100 OPERATING INSTRUCTIONS

MULTI-POP OPERATING INSTRUCTIONS

1. Paper-tape version

The system is distributed as a set of numbered SCB paper-tapes, plus a small 'tail' tape.

(a) Load the SCB tapes under Initial Instructions in numerical order. The final tape will trigger to the **MES: routine of NICE.

(b) Load the 'tail' in Reader 1.

(c) Continue as from (c) in the instructions for the disc system.

2. Magnetic-tape version

The system is distributed as a core dump under an SBL1 lable on a magnetic tape in the standard Elliott format, and may be packed either at 200 bpi or at 556 bpi, this being determined by its labelling. A combined paper-tape trigger and 'tail' is also supplied.

(a) Load the magnetic tape on handler 0.

(b) Input the trigger under Initial Instructions. The dump will be input to store and control triggered to the **MES: routine of NICE. At this point, a program DUMPOP is available, which when entered, will cause the core to be redumped as the last SBL1 dump on a magnetic tape on handler 0, or, if given a parameter, e.g. DUMPOP, <n>., will cause the nth dump to be overwritten by the core, making this the last one on the tape. A trigger is output. (cf DMPALG).

(c) Continue as from (c) in the instructions for the disc system.

3. Disc version

The system is distributed as a core image on tracks 4 to 7 inclusive on a disc pack. Also supplied is a combined paper-tape trigger and 'tail'.

(a) Load the disc pack on handler specified, and other packs for users' files.

(b) Input the trigger under Initial Instructions. The dump is input from the disc and control triggered to the **MES: routine of NICE.

(c) Clear all sense keys and type POP2. If monitor output is required, leave sense key 1 set.

The tail will be read, and after a short delay, the following sequence of messages is output. Note that all input should be terminated by carriage-return, and that the standard line-editing of POP-2 (← and !) can be used to correct errors.

MULTI-POP SYSTEM. ISSUE <issue no.>

USERS: type in the maximum number of channels to be connected to the system during this run (NOTE: this restricts the use to channels 0 to <n-1> during this run, where <n> is the figure typed).

TIME: type in the real time as a four digit integer, e.g. 1430

DATE: type in the date as three separate items separated by spaces or new lines, e.g. 15 NOV 1969

NEWS: whatever is input here will be output to all consoles during this run, at logon time. If no message is required a full stop should be typed, otherwise a message, enclosed in string-quotes (' and `), should be input, e.g. 'SYSTEM RUNNING UNTIL 1730` or .

<time> SYSTEM RUNNING

is now output, and channels 0 to <n-1> as above will now be active and users may log on (see page 8).

It should be noted that apart from the initial loading sequence, NICE software is not used by this system, and after the tail has been read, it is not possible to return to NICE control. Further, pressing the MESSAGE Key will kill the POP-2 system (except in Uni-POP where it is used as the interrupt key). The control teletype is used only as a message displaying device during Multi-POP, any input causes a ← to be output. BELL (CTRL and G) should not be typed.

In disc systems, the disc must not be unloaded or switched off during a run as any subsequent disc transfers will fail.

MESSAGES OUTPUT TO CONTROL TELETYPE DURING MULTI-POP

Messages are of two forms, either informative, or requiring some action on the part of the operators. All messages are output on a new-line and are preceded by the time, if this has changed since the last message, and the name of the user initiating the message. These two items are not included in the descriptions below, the user's name being represented by <name>. The control teletype's bell is run to draw attention to action output.

a) LOGON <c> - <b> BLOCKS, FREE <f>

Indicates that <name> has logged on, on channel <c>, for <b> blocks. There are <f> blocks of store still free, (a block being 512 words).

b) LOGOFF <c> - CPUT <t> - FREE <f>

Indicates that <name> has logged off channel <c>, has used <t> units of CPU Time (a unit is the setting of the Real Time Clock), and there are now <f> blocks of store free on the system.

The following messages (c to f) are output directly or indirectly by a user call of POPMESS.

c) <device name>[<file name>]

In the case of input devices, the operator should load the file with the name <file name> on the device <device name>.

In the case of output devices, this is an indication that a file with the given name is being output on this device.

e.g. RDR1 - [TEST PROGRAM]
LPRINTER - [TEST PROGRAM]

d) [CLOSE <device name>]

This indicates a file (input or output) has been successfully transferred, and that the operator should take appropriate action (e.g. unload the reader or punch, or tear off the lineprinter output).

e.g. [CLOSE PUNCH2]

e) [ABANDON <device name>]

The transfer of a file on <device name> has failed. The device should be unloaded.

f) [ERROR <device name>]

The device <device name> is in an error state (e.g. lineprinter in manual, or no paper in punch). The operator should rectify the fault.

g) [KILLED]

This indicates that user <name> is exceeding his store allocation and is about to be logged off.

h) Garbage messages.

When the storage allocation scheme runs out of store, a 'Garbage Collector' is called to recover free store. This causes output on the control-console in the following format :-

<cell size> 1 2 2 3 <total core in use><total function space in use>

<cell size> is the size (in number of locations) of the cell currently being allocated.

<total core in use> is the amount of store currently being used by the system and users (after the garbage collection).

<total function space in use> is the amount of system and user core which is occupied by functions and closures.

The figures 1 2 2 3 indicate the progress of the garbage collector. No explanation is given here, and in some systems these figures may not be output.

If a garbage message is output in any other format, an addressing error has occurred, it may recover, but the system is likely to crunch!

If garbage collection fails to find enough space, COLLAPSE is entered. It is also entered if 10 garbage collections have occurred since the last collapse. The message

COLLAPSE 123456

is output on the control console. The figures indicate the progress of the collapse.

If the storage allocation scheme fails to find a cell even after a collapse, it checks to see if any user is exceeding his store.

If there is, this user is 'KILLED', and the cell is searched for again. The operator is also in a position to 'KILL' such a user, if sense key 5 is set, the user exceeding his store allocation by the largest amount will be logged off after the next garbage collection.

DISC SYSTEM - HOW TO MAKE DISC TRACKS AVAILABLE FOR USERS TO WRITE TO

The system allows users to read from any part of the disc, but only to write to those tracks which have been allocated to him. This section is intended to show how a disc track can be allocated for use by a particular user, the disc facilities themselves being described on page 33.

A disc file of the following format exists on the LIBRARY tracks :-

[<user name><track No.><track No.>...]
[<user name><track No.>...]

where <user's name> is the name as given by the user when logging on to the system and <track No.> is the number of the disc track being allocated to him. Note that a user can be allocated any number of tracks, and that any track may be allocated to any number of different users.

An example is :-

[RDD 20 21 22]
[JM 21 30 37]

When the system is initialised this file is read into core and is used to decide whether a transfer to disc is legal.

A method is available which will allow a user to write to any track, and is normally used when this file is to be altered.

HOW TO UPDATE THE LIBRARY SYSTEM STORED ON DISC

Certain of the disc tracks are reserved for library programs. These are filed under the disc filing system with DISCSTART set to 3, but are accessed by LIBRARY in a different way.

There is a fixed file held in the library called [LIB DIRECTORY]. This contains a directory for the starts of all the files in the library. It is updated by compiling the library program [LIB CONSDIR] which scans the EASYFILE directories for all the library tracks and writes the new library directory back to disc.

Hence a call of library results in this fixed directory being searched.

A new library program can then be added during a MULTIPOP session by creating the library file and updating the directory.

A listing of the file directories for the library tracks will be supplied with every disc system, and from this it can be seen which of the above tracks are actually in use (at the time of writing only tracks 94 - 99 were occupied). Further tracks can be added as and when the need arises by preparing a tape in the format given below and inserting it at the beginning of the tail as described above (it can go before or after the DISCUSERS list).

e.g. to add track 93 to the library tracks :-

```
F135 'LIBTRACKS KC 'CONS KCE 'LIBTRACKS KA
```

(note 135 is 93 in octal)

This is equivalent to 93::LIBTRACKS->LIBTRACKS; in POP-2.

NOTE: Tracks should not be added to LIBTRACKS in this way unless they have previously been initialized by EASYFILE.

UNI-POP OPERATING INSTRUCTIONS

To load Uni-POP, proceed as for the appropriate system in Multi-POP, up to the point where the 'tail' is read, when the following sequence of messages will be output. Note that all input should be terminated by carriage-return/line-feed, and that the standard line-editing of POP-2 (← and !) can be used to correct errors.

UNI-POP SYSTEM. ISSUE <issue number>

TIME: type in the real time as a four digit integer, e.g. 1430

DATE: type in the date as three items separated by spaces or new lines, e.g. 15 NOV 1969

NAME: type in your user name, e.g. RDD, this is used on line-printer headings, and is required for disc track usage. If a batch system is to be run, type BATCH

SETPOP: is now output and the system is ready for use. NOTE that the message-key should be used as the break key as opposed to CTRL and G in Multi-POP.

CONSOLE MESSAGES IN UNI-POP

The only messages output to the console by the operating system are those giving device load/unload commands initiated by a POPMESS, and are similar to those in Multi-POP except that the time and user's name are omitted, and asterisks are output to distinguish them,

e.g. **LPRINTER[FILE ONE]**

UNI-POP BATCH PROCESSING

At any time while using Uni-POP, a batch operating system may be initiated by typing BATCH;, the specification is as follows :-

The standard input device is set to be reader 1, thus a program to be run should be loaded in this reader. CHARIN will take its characters from this device.

The standard output device is set to be the lineprinter (LP120), thus any characters printed when CUCHAROUT is set to be CHAROUT will be output to this device. The job file heading on the lineprinter will be the file-name of the program tape loaded, if there is one, otherwise it will be [POP BATCH JOB]. Also included in the file heading is the time and date of the run. If a file-name is used, it must be followed by the user's name, if this is omitted, the first item on the program will be taken as the name and errors will occur.

Note that only 'erase' or 'run-out' are allowable characters after the halt code on paper tapes unless the tape has been produced by the system (see PTOUT).

Any POPMESS occurring in the program being compiled will cause an appropriate message to be printed on the control teletype, e.g. **RDR2 [RDD TEST PROGRAM]**. It should be noted that reader 1 and the lineprinter will not be available in this way, and that during batch operation the control teletype is not available as an input/output device.

A standard function POSTMORTEM is entered after any error (i.e. call of ERRFUN). Normally this function is set to be CARRYON, but may be redefined in the batch program.

A halt-code being read on reader 1 will automatically terminate that program, all current devices are closed, the core is cleared, and the message :-

JOB FINISHED. LOAD NEXT (a)

is output. The next program to be run should be loaded in reader 1.

A standard function ABANJOB is available for use by programs running under the batch. When called, ABANJOB causes that job to be killed and has the same effect as if the halt-code had been read on the end of the tape.

A batch job may be terminated at any time by pressing the message key. This causes the store to be cleared, and the message :

JOB ABANDONED. LOAD NEXT (b)

to be output.

To terminate the batch and return to on-line mode, the message key should be pressed while reader 1 is waiting for the next program to be loaded, i.e. immediately after either message (a) or (b) above. The message :-

BATCH STREAM TERMINATED

NAME:

is output, and the system is back to normal console input/output, waiting for a user's name to be input.

INDEX TO ALL POP-2 WORDS

This index lists all standard functions, variables, and syntax words specified in both the Reference Manual and this manual. Each word is followed by a letter which denotes its type and which have the following meanings :-

F denotes a Function
M denotes a Macro
O denotes an Operation
S denotes a Syntax word
V denotes a General Variable

The page numbers relating to this manual are given, and R denotes that a description is also given in the Reference Manual.