

AUGUST 1974

MUSS BASIC LIBRARY MANUAL
VOLUME 2

UNIVERSITY OF MANCHESTER
DEPARTMENT OF COMPUTER SCIENCE

080
NSC I

Preface

This manual describes the library indexing systems and the 'compatible' basic libraries of the MUSS systems. Users of the 1900 system should take note of the appendix which lists the unavoidable divergences of that system. The basic library consists of the general housekeeping procedures for starting, stopping and trapping processes, the I/O procedures and the organizational commands which run in privileged mode to provide an interface with the operating system.

CONTENTS

CHAPTER 1	The Organisation of the Library System	1.1
	Introduction	1.1
	The Common Directory	1.1
	CTL Coding For Common Procedure Properties	1.5
CHAPTER 2	Starting, Stopping and Trapping a Process	2.1
	Program Control Sequence	2.2
CHAPTER 3	The General Organisation of Input and Output	3.1
CHAPTER 4	The Library Procedures	4.1
	The Mode Word	4.1
	Input Procedures	4.3
	Output Procedures	4.11
	Algol-like Input/Output Procedures	4.22
	File Manipulating Procedures	4.38
	File Disc Procedures	4.40
	System Performance Monitor Procedures	4.43
	I.O Fault Numbers	4.67
CHAPTER 5	Organisational Commands	5.1
	Organisational Command Fault Numbers	5.2
	Process Organisational Commands	5.4
	Virtual Store Organisational Commands	5.21
	File Organisational Commands	5.33
	Communication Organisational Commands	5.46
	User Management Organisational Commands	5.58
	Disc Commands	5.65
APPENDIX	The 1900 Library	

1944

1945

1946

1947

1948

1949

1950

1951

1952

1953

1954

1955

1956

1957

1958

1959

1960

Access to library procedures is via a set of tables known as the common directory. Some library procedures exist which can be used to access the directory. Although initially the common directory will be a simple structure some attempt will be made to improve look up times as its size grows.

The Common Directory

The common directory consists of three main tables or lists which contain information about the common procedures.

The three tables are each resident in a common segment and are:-

1. The Common Procedure Name List (CPNL) - S8211
 2. The Common Procedure Entry Table (CPET) - S8221
 3. The Common Procedure Property Table (CPPT) - S8222
- Segments 8223 onwards contain the procedures.

The layout of each table is shown on the following page.

The namelist is structured into two levels. The second level

is a simple linear string of items, each item consisting of the following:-

1. An index into PET (16 bits)

2. A character count for the procedure name (8 bits)

3. The characters of the name

The items in the second level are alphabetically sorted on their

names.

The first level consists of entries which index into the second level. Each entry consists of:-

1. First four chars of name contained in second level

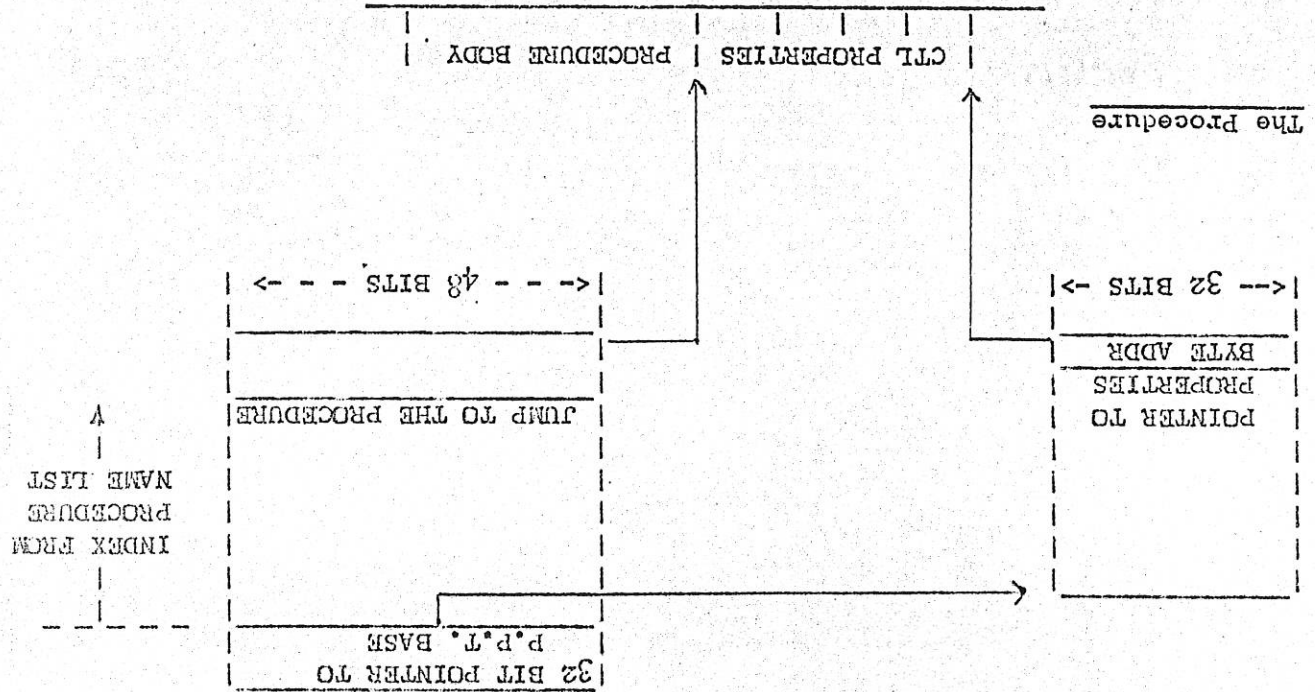
2. Index into second level

An entry appears in the first level for at least every third

name in the second level.

The procedure entry table is a table of 48 bit entries which are instructions to jump to the first instruction of the common procedure. For Organisational commands the table contains the XC6 <OPERAND> instruction which enters the command. Entry zero is a 32-bit pointer to the procedure property table. In general, slots in the procedure entry table (and PRT) will be assigned once only as each common procedure is first created. The first slot is used to contain the entry point for the START routine.

The procedure property table points to the vector of properties which precede the procedure body. This table will be used mainly for the handling of non-local jumps and parametric procedures as required by the CTL. The properties themselves are fully defined in the COMPILER WRITERS MANUAL pages 95 to 98.



Procedure Entry Table

Procedure Property Table

256	16	8	n	16	8	n	16	8	n	16	8	n
INDEX	CHR	NAME CHARS	INFO PRT COUNT									
LEVEL 2 OF PROCEDURE NAME LIST												
0	32	32	32									
FIRST 4 CHARS	OF NAME IN	2nd LEVEL	ITEM	ADDRESS OF	2nd LEVEL							
LEVEL 1 OF PROCEDURE NAME LIST												

Procedure Name List

The Common Directory

17.3.72

The procedure call will then stack the index to the procedure property table and jump (via a literal jump) to the procedure. The second usage is by the fault monitor routine or the non local jump sequence which uses the stacked value of the index into the PET to determine the properties of the procedure.

STKLINK 11
STACK PPT.INDEX
STACK PARAMETERS
JUMP TO.PET.ENTRY

The typical usage made of the common directory is twofold. During the compilation of a program the CTL will plant calls for the common procedures using the FIND.N procedure to find the index of the common procedure in PET from the procedure name. A typical calling sequence planted by the CTL will be:-

The CTL Properties for Common Procedures

This is a vector of 32-bit elements which contains the

following information:-

1. The entry address of the procedure
2. The status word

The status word contains four 8-bit quantities which are decoded

as follows:-

1. (MSE) number of parameters
2. The mode of result
3. The procedure type
4. The textual level (one for common)

If the number of parameters are zero then this is all that exists, however if there are parameters then the next few elements of the vector contain mode information about each parameter.

3. The index into this vector of the first element containing the parameter modes (usually 3)
4. The mode of each parameter (8 bits per parameter)

The procedure type and the mode bits are defined below but can also be found in the Compiler Writers Manual, pages 97 and 40.

0	=>	proc
1	=>	subproc
2	=>	static
3	=>	variable parameter
4	=>	block
1	=>	common proc
0	=>	not a common proc

bit 4

bits 5 - 7

The procedure type is decoded:-

0	=>	1 bit
5	=>	32 bits
6	=>	64 bits
7	=>	128 bits

bits 5 - 7 are the SIZE

1	=>	Boolean
2	=>	fixed signed
3	=>	fixed unsigned
4	=>	floating
5	=>	decimal
7	=>	descriptor
9	=>	Procedure

bits 1 - 4 are the TYPE

bit 0 - spare

NOTE Bit 7 is the least significant bit.

The mode bits decode:-

bit 0	1	=>	stacked results (last param property gives)
	0	=>	no stacked results (number of results)
bit 1	1	=>	address of properties present
	0	=>	not present
bit 2	1	=>	postlude set
	0	=>	not set
bit 3	1	=>	an executive proc
	0	=>	not an executive proc

This procedure is entered by jumping to the sixtieth entry in the Procedure Entry table. The input parameter is a string descriptor pointing to the procedure name. On exit from the procedure the register X will be set to zero if the procedure could not be found otherwise it is set to the index into CPWT.

The LOOK.UP.COMMON.PROC procedure (FIND.N)

ADD.N

ADD.COMMON.PROC (S, S)

This will add the name and properties of the procedure whose name is described by P1 into the Common Directory. The code of the procedure has to be added to the system separately.

P1 String descriptor to the characters of the name of the procedure
P2 Descriptor to the procedure's property table (32-bit entries)

Introduction

A few library procedures are concerned with controlling the execution of a process, starting it, terminating it and ensuring that any output produced is sent to its destination should the process terminate involuntarily.

The software trapping mechanism is included in this set of procedures and allows a user to enter his own procedures should any traps occur. Library procedures which store/restore the CPU registers are provided. These enable software trap procedures to be written in a machine independent language.

START () [The procedure name]

START [The flowchart name]

This is a common procedure which is called to start a process. It assumes a program control sequence will be found on input stream 0 and if either input 0 or output 0 have not been defined then it will define them. The process program control sequence procedure is then called to process input 0 and the exit is to the STOP procedure.

STOP (1)

STOP

This procedure is used to terminate oneself. All the outputs are ended and logging information is printed upon the lowest output stream defined as other than a file (if any). The process is terminated by calling TERMINATE.PROCESS with P1 as the reason.

The program control sequence is not part of the program proper and is language independent. It is interpreted as follows:-

First, it saves the current traps, and sets the traps to the standard trap routine. Secondly, it selects the input stream specified by the parameter. Then it enters a loop which looks at the first two symbols of the next line of the selected input. If these symbols are ** it interprets the line as a call for a common library procedure, else the traps on entry are reset and the program control sequence is exited.

If control is returned from a common procedure the loop will be resumed unless PWA is non zero in which case trap 4 is entered. If the compilers do this it means that the library procedures can be provided for dumping compiled programs etc. which are available to users of all languages.

It is expected that the most frequently used common procedures in program control sequences will be SET INPUT, SET OUTPUT and the compilers.

A call for a common procedure will take the form:-

** <NAME> (<PARAMS>)

where

<PARAMS> ::= <PARAM> | <PARAMS> , <PARAM>
 <PARAM> ::= <INTEGER> | <CH.LIT> | <CH.STRING> | <HEX.LIT>
 <INTEGER> ::= [+ | - | NIL] <NUMBER>
 <CH.LIT> ::= $\frac{1}{2}$ <CHRS>
 <CH.STRING> ::= $\frac{1}{2}$ <CHRS>
 <HEX.LIT> ::= $\frac{1}{2}$ <HEX.DIGITS>
 <HEX.DIGITS> ::= [<HEX> | <HEX> (INTEGER)] [<HEX.DIGITS> | NIL]
 <CHRS> ::= <CHRS> <CHR.HEX> | <CHR.HEX>
 <CHR.HEX> ::= ANY UK ISO CHR | <HEX.PAIR>
 <HEX.PAIR> ::= <VB> <HEX> <HEX> <VB>
 <HEX> ::= Any hexadecimal digit
 <VB> ::= |

P1 the stream number

ENTER.TRAP (I, I)

This procedure causes an entry to be made to the software trap which is specified in parameter P1. The reason for the entry is contained in parameter P2 and these two parameters will be left on the stack as parameters for the trap processing procedure. This procedure can be set by the user using SET.TRAP.ADDRESS.

P1 the trap number
P2 the reason

SET.TRAP (I, P)

SET.TRAP

This procedure arranges that the procedure (parameter P2) is entered when the trap (P1) is invoked. The procedure (P2) should have two parameters which will be set to the trap number and the reason. No registers will be altered between the call of ENTER.TRAP and actually entering the trap procedure.

P1 the trap number
P2 the procedure (This is a 128 bit quantity consisting of two descriptors, the top 64 bits are for the display and the bottom 64 bits are for the properties).

READ.TRAP (I) P

This returns the procedure parameter which was used in SET.TRAP. The result is returned in PWW1 and PWW2.

P1 the trap number

INIT.T ()

This resets all trap addresses to their default values.

P1 This is a vector descriptor (of 64 bit elements) specifying the 'address' of the register dump.

This procedure loads the CPU registers (see DUMP.REGISTERS) from the area defined by P1.

LOAD.REGISTERS (S)

P1 This is a vector descriptor (of 64 bit elements) specifying the 'address' for the register dump.

64	D
	BOD
	DOD
	X
	AOD
	A
	XD
	AFX
	XNB
	B

This procedure saves the CPU register in the area defined by P1. For MVS the format of the register dump is as follows:-

DUMP.REGISTERS (S)

0	-	Program Fault
1	-	Limit/external interrupt
2	-	Local timer
3	-	Message
4	-	I/O FAULTS
5	-	System Performance Monitor
6	-	Compiler Faults
7	-	User Programs

ENTER ()

This procedure is called to enter a compiled program. It assumes the compiler has left two parameters in fixed locations in segment 0 (words 128 and 130 on the 1900).

128	ENTER switch, 0 => do not enter
130	Segment number in which code is to be executed

On MVS, the ENTER switch is 'FAULTY' (word 63) and contains the number of faults (-1) negated for a faulty program, i.e., =< 0 => do not enter. Word 53 contains the address (in 16 bit units) at which the code is to be executed.

If the 'ENTER' switch is positive the code segment is entered at line 0. If the 'ENTER' switch is =< zero, the procedure returns with TWO non zero.

EXIT (1,1)

This procedure facilitates exit from procedures which have been called using CALL.

P1	is the segment number of a segment to be released
P2	is the address to jump to after releasing the segment.

This procedure consists of LOAD, followed by ENTER. The parameters are as for LOAD.

CALL (U64,I,I,S)

This procedure recreates a program which was filled by DEFINE. The first parameter gives the file name. The remaining 3 are the parameters for the loaded program. If LOAD is called from the program control sequence, only those parameters which are actually required need be written; the remainder will be set zero by the program control sequence. If the loaded program is to be entered by calling ENTER, and it requires parameters, there should be no orders between the calls of LOAD and ENTER which involve stacking.

LOAD (U64,I,I,S)

→ + 4 (32 bit instruction)
32 bits to be used as a control word

64 bits of the code segment should be:-
as no CTL areas will be associated with it. P2 should be zero. The first

For an XPL program, only the code compiled is defined as a file, lists, if this is specified in P2.
segment filled with the code is that containing the CTL name and properly the line table is also included. If Perm is being compiled, the only are included in the filled program segment. If the next bit of P2 is set, the CTL name and properly lists, as required for runtime fault monitoring information are copied to this segment. If the bottom bit of P2 is set, control information in the code segment, and those which already contain filling the program, all CTL areas associated with it are recorded in the. Before the program is defined as the file named in P1, and not entered. Before This procedure is similar to ENTER, but, if FAULTY is positive.

DEFINE (U64,I)

The Input/Output ProceduresGeneral Organisation

The input/output procedures require three 64-bit words of information in segment 0 and a control segment (31) containing a control area for each of the 8 input and output streams and some additional workspace. Facilities exist for a supervisor to set up this control segment and pass it together with some I/O streams to a process which it is creating. Thus some I/O streams might exist within the process before it starts. Normally only special on-line supervisors will act in this way, and the START procedure will set up a control segment. Unless they already exist START will also define an input 0 to be mode 0 and an output 0 to be mode 0 destination LPT5E. In either case the START procedure will set the trap vector to the address of the TRAP procedure and will select input 0 and output 0. It will then attempt to read and obey a job control sequence from stream 0.

The information kept in segment 0 by the input/output procedures is:-

- 1) a descriptor INPUT pointing to the last symbol read in the current stream.
- 2) a descriptor OUTPUT pointing to the next symbol position in the current stream.
- 3) a 32-bit pointer IN.CONTROL pointing to the control area of the current input stream.
- 4) a 32-bit pointer OUT.CONTROL pointing to the control area of the current output stream.

The control segment has 8 input sections and 8 output sections. The inputs correspond to the 8 message channels. An input section contains the following:-

1. The current value of the input descriptor for this stream (in the case of the currently selected stream, it may not be up to date)

2. The address of the source of the input (as given by the read message command)

3. The short message buffer (containing the device mode, type cost used and the message itself)

4. An EOL to trap those cases where the short message didn't have one

5. The segment number containing the input text (-1 if no segment exists)

6. The mode of input.

The output section contains a similar set of quantities as well as some area at the start of the segment of output when there is one.

1. The current value of the output descriptor

2. The address of the destination of the stream (to be

given in bond message)

3. The short message or file information

4. An EOF. The output pointer is set to point before this

when the input/output is initialised.

5. The segment containing the output text (-1 if no segment)

6. The mode.

The accompanying segment of text contains a few control quantities which are used to limit the output used by the user.

1. The no of times the output segment has been broken

2. The maximum no of times the segment can be broken

3. The maximum no of newlines allowed (the no of

newlines used is accumulated in the short message)

4. The actual text.

The Input and Output Control Segment

3.4

| STREAM 1 |
| 16 TRAP ENTRIES | 40 WORDS ORG. ORDER WORK SPACE | INPUT | OUTPUT | IN | OUT |

| STREAM 7 |
| IN | OUT | PPC. SEQ WORK SPACE |

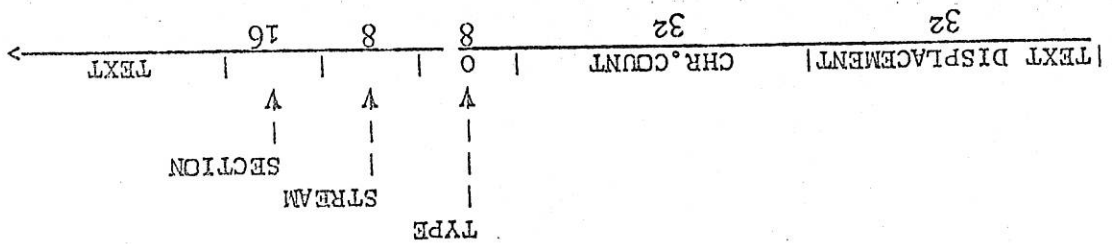
SEGMENT 8191 (31)

17.3.72

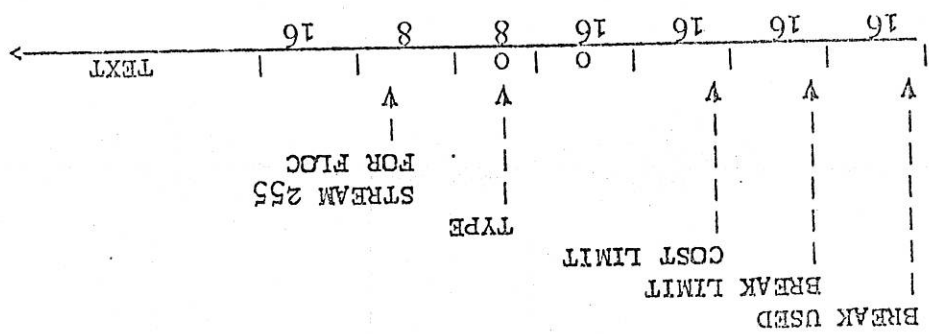
[illegible][illegible]

The Text Segment Layout

Output Segment



During the accumulation of output the layout will be:-



If necessary user procedures may access these areas themselves although this is discouraged unless the user has an intimate knowledge of the flow diagrams and code of the basic input and output.

The I/O Library Procs

Chapter 4

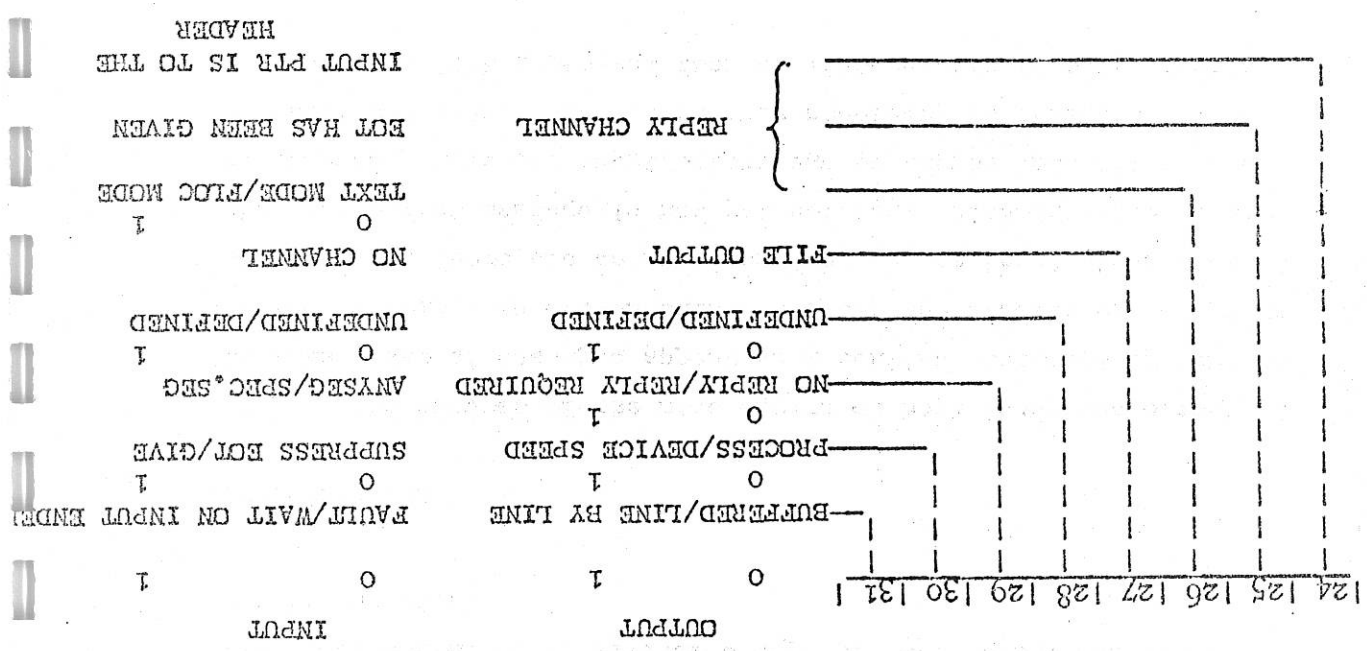
Introduction

The following pages contain a description of the basic input and output procedures and their specifications. The implementation details concern the MVS implementation but the MX/3 implementation is similar except for the points mentioned in the Appendix.

Mode Settings

The purpose of the mode digits on both input and output is to make a set of messages appear as a continuous stream of symbols to the program. An option exists whereby the process can elect to be halted if there are no more messages on the input channel or if the last output message is not yet printed. Alternatively it can be trapped. Also the input/output can be mapped into either long or short messages. Thus it is only necessary to change the mode setting to switch a program from on-line running to batch running.

NOTE: For output stream, if the mode is for a file output with reply required, then the file is for a private disc.



Input procedures

IN.SPEC(I, I, I)

IN.SPK

This procedure defines the nature of the stream specified. The control area for this stream is set from the input parameters. In addition the MODE is set as though an EOT had just been read and the input pointer is set to point to the EOT in the header. This is so that when IN.SYM is first called the first message on the associated channel will be the start of the input stream. If the mode is specified as WAIT then the input channel status will be set 'OPEN TO ACC' otherwise no action will be taken. The procedure has no effect if the stream is already defined unless bit 29 of P2 is set. In this case the stream will be set up again to conform to the new specification. If there is a partially read message it will be lost.

P1	Input stream number
P2	The type of input (MODE)

BIT

31 FAULT/WAIT ON INPUT ENDED

30 SUPPRESS/GIVE EOT

29 OVERRIDE PREVIOUS IN.SPEC

25 to 28 USED BY THE I.O procedures

24

START READING EVERY NEW MESSAGE FROM THE

HEADER

P3 The segment in which the input text is to be put (if any).

SET.INPUT(I, I)

SET.INP

This is as for IN.SPEC with P3 = -1.

P1 As for IN.SPEC

P2 As for IN.SPEC

SELECT.INPUT(I)

SEL.INP

This stores the current input pointer from segment zero to the control segment and replaces both the input pointer and the INPUT control word by the values for the stream specified. If the stream has not been defined then a fault is indicated.

CUR.INPUT

CURRENT.INPUT(I)

The current input stream is yielded as a result in the

internal register X.

I.POS

I.POS(I)

A 32-bit pointer to the remainder of the input stream is

yielded as a result in X.

S.I.POS

SET.I.POS(I)

The origin field of the INPUT descriptor will be set to P1.

P1 A position in the stream previously given by I.POS.

SEL.TEXT

SELECT.TEXT()

This sets the input pointer in segment zero to point to the

start of the input text contained within the segment and sets the

page/line count to 1/1. If no segment exists then it is set pointing

to the short message. The command only works on the current message

It should not be used on a stream before the first message has been

read.

SEL.HEAD

SELECT.HEADER()

This sets the input pointer in segment zero to point to

the start of the current short message, after the device mode and

cost information. It should not be used on a stream before the

first message has been read.

IN.SYM

IN.SYM() I

This procedure yields as a result in the register X the next symbol on the selected input stream. If the stream is undefined then the user is faulted. The action taken upon reading an EOT depends upon the setting of the MODE (see IN.SPEC).

The normal path through the procedure is quite simple, consisting of accessing the next character and testing to see if it is a control character before yielding it as the result. Additional action is required for LF, FF and EOT. For each LF the line count is incremented by 1, similarly for each FF the page count is incremented by 1. If the symbol is an EOT then a complex subroutine is called to determine the necessary action. This routine is also used by NEXT.SYM.

The effects of reading beyond an EOT are summarized below.

1. READING FROM TEXT (OR HEADER)

The EOT symbol is given depending upon the setting of the MODE and the next IN.SYM will result in one of two actions. If there are messages waiting on the channel then the first of these is read, the input pointers are reset to either the start of the text or the header, depending on Bit 24 of the mode and the first symbol is yielded as the result. If no messages are waiting then the user is either faulted or suspended awaiting the arrival of another message depending upon the setting of MODE.

2. READING FROM A FILE INPUT

The EOT symbol is given and then the user is faulted if he reads again.

3. READING FROM A CREATED INPUT

As for file input.

I.LINE ()

This procedure yields as a result in X the packed page/line count (16 bits each).

DEL.INP

DELETE.INPUT(I)

This procedure creates the effect of reading up to the EOT symbol on the specified stream (and including it if bit 1 of mode is 1) and deleting the segment (if any). The next call of IN.SYM for this stream will attempt to read the first symbol of the next message and the action taken will be according to the setting of the MODE.

P1 The stream of input to be deleted.

END.INP

END.INPUT(I)

This procedure cancels the specified stream of input releasing any segment and marking the stream as undefined.

The channel will be closed.

The procedure sets the MODE of the specified channel to undefined. The input pointer for the specified channel is

set to point to the EOT of the end of the header.

P1 The stream of input to be ended

CRE.INP

CREATE.INPUT(I, S)

The string of characters pointed to by P2 is used to form the input of the specified stream (P1). If the stream already exists then the trap is entered.

P1 is the stream to be used as the input

P2 is a descriptor of the text to be used on input

IN.SEG.SPEC (I, I)

The segment specified is set up as the input text and the MODE is set to give EOT, NO CHANNEL. If the stream is already defined the command is ignored.

P1 The stream number
P2 The segment

IN.FILE.SPEC(I, U64, I) I.F.SPK

This procedure causes the specified file to be used as input. It is assumed to be in the standard text segment format. MODE is set to give EOT, FILE input. If the stream is already defined the command is ignored.

P1 is the input stream number
P2 is the name of the file (8 characters)
P3 is the segment number of the segment which will hold the file

SET.FILE.INPUT(I, U64) S.F.I

This sets the segment number to hold the file to -1 and calls IN.FILE.SPEC.

P1 is the input stream number
P2 is the name of the file (8 characters)

SET.FILE.IO (I, I, U64, I)

This procedure causes a file to be used as input, and sets up an output stream which will eventually write to the file (using FILE). The file is opened with read permission, and exclusive access. If either stream is already defined the trap is entered.

P1 is the input stream number
P2 is the output stream number
P3 is the file name (8 characters)
P4 is the cost

IN·STAT

IN·STATUS (I) I

This procedure copies the mode bits and the segment number into the register X. They are stored as 'SEG <- 8 + MODE'.

P1 is the input stream number

IN·SOURCE(I, S)

This procedure supplies information about the source of the specified input stream. The output parameters are copied from the relevant areas in the control segment.

P1 is the input stream number

P2 is a descriptor of vector of at least six

32-bit elements into which is put:-

SPN

PROCESS ID

REPLY CHANNEL

USER ID

DEVICE NUMBER

TYPE

A trap is entered if the stream is undefined or was defined by SET·FILE.

INPUT/IN·FILE·SPEC or CREATE·INPUT.

Note that this procedure can be used to determine whether an input stream is online or offline, if offline the reply channel will be negative.

IN·REPLY(I, I, I, I)

IN·REP

This procedure sets up an input stream which is to be used

to receive replies generated when the specified output stream is broken. If the output stream is not defined or if the input is

defined then a trap is entered. The reply channel and reply

required bits are set in the mode for the output stream. The

channel status is set dedicated to the process which is specified as the destination in the output stream.

P1 is the input stream number

P2 is the mode (as for IN·SPEC)

P3 is the segment number to contain the input text (if any)

NXT.SYM

NEXT.SYM(I)

This procedure has the same effect as IN.SYM except that the input pointer is not advanced to the next symbol. The result is given in the register X.

IN.I32

IN.I(I)

This reads in a 32-bit signed integer from the currently selected input stream returning the value in X. A trap is entered if no integer is found.

IN.U64(U64)

This reads in up to 16 hexadecimal digits from the currently selected input stream returning the value in A. (<INTEGER>) after a hexadecimal digit repeats the digit.

IN.C17(U64)

This reads in up to 8 characters enclosed in double quotes from the currently selected input stream, returning the string by enclosing 2 hexadecimal digits in exclamation marks.

IN.CSTR(S)S

This reads in a character string enclosed in double quotes from the currently selected input stream, returning in D a string descriptor pointing at the string. Non printable characters may be represented by enclosing 2 hexadecimal digits in exclamation marks.

P1 is a descriptor to a buffer to be used to store the string.

IN.R()R

This reads a 64-bit floating point number from the currently selected input stream, returning in A the number coded in form as shown in Basic Programming Manual, 4.3.2(d).

The input form must be:-

<SIGNED.INTEGER>[.<INTEGER>|<NIL>][@<SIGNED.INTEGER>|<NIL>]

where the integer following the @ is a decimal exponent.

IN.BACK.SPACE ()

IN.BK.SP

This will move the input pointer back one character, provided that this action will not cause it to be moved out of the start of the buffer. If it would, then it is left where it is and trap 4 is entered with reason 20.

Output procedures

OUT.SPEC(I, UG4, I, S, I, I, I, I, I, I)

OUT.SPK

This procedure specifies the nature of an output stream. The parameters determine the destination and type of the output. The destination is given as the name of the receiving process. The type of output is specified by setting the MODE parameter. If the output is to be accumulated in a segment then the TITLE given will be attached to the device and cost to form the short message which accompanies the segment. If the output is line by line then the TITLE is ignored. The procedure sets the output pointer in the control area to point to the short message/segment and the base of the segment (if any) which is initialised into the standard text segment format. A few extra words at the base of the segment are also used for holding some costing information.

P1	is the output stream number
P2	is the name of the process to which the output will be sent, rules for name as in LOOK UP PROCESS.
P3	is the cost limit (in newlines)
P4	is a descriptor specifying a title string. If this is longer than the short message a fault is generated. If the origin field of this descriptor is negative, a default title of ***A USERNAME PASSWORD PROCESS NAME will be inserted.
P5	Channel number of destination
P6	The device number
P7	The device mode
P8	The segment number of the segment in which the output will be accumulated
P9	The number of times the output is to be broken
P10	The Mode of output '0' '1'

31	buffered/line by line
30	process/device speed
29	OVERRIDE PREVIOUS OUT.SPECS

SET.OUTPUT

SET.OUTPUT(I, U64, I)

This procedure stacks suitable values for the parameters P4, P5, P6, P7, P8, P9 & P10 of OUT.SPEC and then calls OUT.SPEC. The parameters are set as follows:-

P4 A descriptor to the string ***A USER PASSWORD
PROCESS NAME

P5 the channel no. is set to zero

P6,P7 these are set to zero to denote 'Any' device

P8 set to -1

P9 set to 1

P10 set to buffered and at process speed (i.e. zero)

OUT.FILE.SPEC(I, U64, I, I) O.F.SPK

This procedure specifies that this output stream is to

become a file when a BREAK.OUTPUT is done on this stream. The

file name specified is copied into the short message in the

output control area. The command is ignored if the stream already

exists.

P1 is the output stream number

P2 is the file name (8 characters)

P3 is the cost limit (in newlines)

P4 is the segment number of the segment which

will be used to contain the output

SET.FILE.OUTPUT(I, U64, I) S.F.O

This procedure sets parameter P4 of OUT.FILE.SPEC to a suitable value and then calls OUT.FILE.SPEC. The value chosen

is:-

P4 is set to -1

SEL.OUT

SELECT.OUTPUT(I)

This procedure arranges that OUT.SYM will subsequently

operate on the stream (P1). The value of the present output

pointer in segment zero is stored in the control area, the new

stream number is used to calculate a new value for OUT.CONTROL

in segment zero and the new output pointer is copied into

segment zero.

P1 is the output stream number

BREAK.OUTPUT()

BREAK.OUT

The procedure sends the accumulated output to its destination.

The size of the segment will be reduced to the minimum before being

sent as a message to its destination. The action taken depends upon

the type of output. An EOT is appended and the following actions

are taken:--

1. LINE BY LINE

The line of output is sent as a short message to the

process and the stream reinitialised.

2. BUFFERED, (created by OUT.SPEC)

The segment and title are sent to the process. If the

number of times this output has been broken is greater

than or equal to the limit specified then the output is

ended (END.OUTPUT) otherwise the stream is re-initialised

and the 'breaks used' is incremented.

3. A FILE OUTPUT

The effect is as for END.OUTPUT

CUR.CUR

The number of the current output stream is yielded as a

result in register X.

O.POS(I)

O.POS

The procedure returns a pointer to the next free character

position on the currently selected output. The result is returned

in X.

SET.O.POS(I)

S.O.POS

The procedure sets the origin field of the current output

pointer in segment zero to the parameter (P1).

P1 is a 32-bit byte addr

DELETE.OUTPUT(I)

DEL.OUT

The procedure deletes the output accumulated on the

specified output stream. The following action is then taken:-

1. LINE BY LINE

The output pointer in the control segment is

reset to the start of the short message.

2. BUFFERED (created by OUT.SPEC)

The output pointer is reset to the beginning of the

segment of text. If the number of times the output has been

broken is greater than or equal to the limit specified then

the action is as for END.OUTPUT. Otherwise the 'breaks used'

is incremented.

3. A FILE OUTPUT

The output segment is released and the control area

for the stream set up as undefined.

END.OUTPUT(1)

END.OUTPUT

The procedure selects the specified stream and sends the output to its destination after first changing the size of the segment to the minimum required and then initialises the output stream to undefined, EOT read etc. No action is taken if the stream does not exist otherwise the action is as follows:-

1. LINE BY LINE

The effect is as for NEWLINE and the line of output is sent as a short message to the process. The following is also sent as a second message to the process:-

***Z

The output control area for this stream is then initialised as undefined.

2. BUFFERED (created by OUT.SPEC)

The output is sent to the process and the output control area for this stream is then initialised as undefined.

3. A FILE OUTPUT

An attempt is made to produce a file of the output segment. The output is first defined as the file, if this fails for any reason except 'file already exists' then the user is faulted. If the file already exists then replace file is attempted and if this fails the user is faulted. The output control area for this stream is then initialised as undefined.

P1 is the output stream number

P1 is the symbol to be output

The effect is as for BREAK.OUTPUT.

BUFFERED

specified destination and the output pointer is reset.

The short message accumulated is sent to the

LINE BY LINE

This causes the parameter to become the next character of the output stream advancing the output pointer on by one character. If the output buffer is full then the following action is taken:-

OUT.SYM

OUT.SYM (1)

This gives, as a result, a pointer in X which points to the previous newline symbol or to the start of the output text if no newline symbol is found. The procedure scans back from the current position until it finds a newline symbol.

LINE.POS

LINE.POS (1)

P1 is the number of newlines required

This produces the required number of newlines. The parameter should be ≥ 0 , ≤ 63 . For buffered mode the effect is the same as repeating OUT.SYM the required number of times except that the cost is used to trap output exceeded. In on-line mode a newline (#12) character is planted and BREAK.OUTPUT is called. This is repeated P1 times.

NEWLINE

NEWLINES (1)

OUT.STATUS

OUT.STATUS (I) I

This gives the segment number and MODE in the register

X. The format is 'SEG <- 8 + MODE'.

P1 is the stream number

SPACES (I) SPACES

This outputs the number of spaces specified by the

parameter. It should be $\geq 0 \leq 127$. The effect is as if OUT.SYM had been repeated the required number of times.

P1 the number of spaces required

CAPTION (S) CAPTION

This copies the string of characters described by the

descriptor into the current output stream. The effect is as

if OUT.SYM had been used to copy the characters.

P1 A descriptor to a string of characters

PROMPT (S)

This is similar to CAPTION, except that the string will only be

printed if there is no further input on the current input stream. It is

used to issue a request for input to an online user. If the current input

stream is offline, PROMPT will have no effect.

ECHO.LINE ()

This procedure will cause the current input line to be printed on

the current output stream if the selected input stream is an offline (buffered) stream. It may be used by editors etc. to echo commands when used in offline

mode. It has no effect in online mode.

P1	the output stream number
P2	the corresponding input stream number
P3	the cost
P4	the title
P5	the segment (if any) which will hold the text
P6	the number of times the output is to be broken
P7	the mode

This defines an output stream which is to be used as a reply to an input stream. The command will be dynamic, sending the output to the process associated with the input stream at the time the output is dispatched. A fault is generated if the input stream is undefined. The channel to which the output is sent is that specified as a reply channel on the input. If no reply was required channel zero is assumed.

OUT.REPLY (I, I, I, S, I, I, I)

OUT.I (I32, I32)

OUT.I32

This prints the signed integer (P1) in decimal format on the currently selected output stream. The number is printed with a field width of P2 characters, preceded by a minus sign if negative. If positive then a space is printed unless P2 is zero when no space is printed. The numbers are printed right justified within the field space and left justified if the number overflows the field.

P2

the field width

P1

the number to be printed

OUT.TIME

OUT.TIME

This prints the time in the following format:-

<SP/DIG><O/DIG>:<SP/DIG><O/DIG><SP/DIG><O/DIG>

where

<SP/DIG> ::= SPACE | 1 | 2 | 3 . . . 7 | 8 | 9

and

<O/DIG> ::= 0 | 1 | 2 | 3 . . . 7 | 8 | 9

and the total format is:-

Hours : minutes : seconds

OUT.DATE

OUT.DATE

This prints the date in a similar format to OUT.TIME,

f.e.,

DAY : MONTH : YEAR

OUT.HEX (U32)

OUT.HEX

This prints the 32 bit number in hexadecimal.

P1

The number to be printed

OUT.STAT

This prints the statistics about the process. Those

printed are:-

STATISTIC	UNIT
TIME, DATE	
COST (BALANCE, LIMIT)	shekels
CPU TIME (BALANCE, LIMIT)	mill seconds
MAXIMUM COST	shekels
EXECUTIVE TIME	mill seconds
WAIT TIME (BALANCE, LIMIT)	seconds
SEGMENTS (USED, LIMIT)	

The format is as below:-

PROCESS STATISTICS ON 22: 2:72 AT 9: 5:26

CPU	COST	MAX EXEC	WAIT	SEGMENTS	BAL	LIM	BAL	LIM	USED	LIM
-----	------	----------	------	----------	-----	-----	-----	-----	------	-----

OUT.F (1,5,1)

This procedure is used to print fault messages. The input parameter (P1) specifies the message to be printed. Numbers -50 - 0 represent the executive procedure faults (See 5.1). Fault numbers > 0 represent the compile time faults for the compilers. (See MVS Compiler Writers Manual, P112). Fault numbers > 200 represent input/output fault messages, (see 4.31). [201 is message for I/O fault 1, etc.]. Fault numbers > 300 represent run time fault messages.

P1	Fault number
P2	Descriptor of message string
P3	Page (top 16 bits) and line number

A page/line number is first output if P3 is non-zero. (If P3 = 0, spaces are output). '00' is then output followed by a string of characters, (P2) if P2 is non-zero. Finally, the fault message is output followed by a newline. The segment zero location, FAULTY, is decremented by one.

OUT.M (I,S,I)

This is similar to OUT.F, but is used to print warning messages. The message is output preceded by '??'. OUTM does not decrement FAULTY.

BEND.OUTPUT ()

BEND.OUT

This ENDS the currently selected output stream inserting the control information in the base of the text segment. It does not send the segment as a message to its destination, but returns the segment number in PW1. A trap is entered if the output is undefined or on line.

CH.DEST (I, S)

This changes the destination of an output stream. The descriptor (P2) should be similar to that given by IN.SOURCE. If the stream does not exist or was not created by OUT.SPMC or SET.OUTPUT then the user is faulted.

P1

the stream number

P2

the new destination (as for IN.SOURCE)

DISPLAY (I)

This procedure writes P1 to the DISPLAY lights on the console.

P1

value to be displayed

12.9.73/5

This will move the output pointer back one character, provided that this action will not cause it to be moved out of the start of the buffer. If it would, then it is left where it is and trap 4 is entered with reason 20.

O.BK.SP

OUT.BACK.SPACE ()

16 - 31 line

Bits 0 - 15 Page

where

P1 Page/line number

This procedure prints the page/line number as specified by P1 on the currently selected output stream. The page and line parts are printed in decimal with a full width of 4 separated by a "...".

OUT.LINE (I)

P1 Address of first word to be dumped
P2 Address of last word to be dumped

This produces a hexadecimal store dump on the currently selected output stream. The parameters are the start and finish addresses of the dump, and must be word addresses.

OUT.STACK (I, I)

OUT.STK

This procedure searches from the current input position for the next number, in a format allowed by the ALGOL report 2.5.1. Preceding spaces, newlines and horizontal tabulation characters are ignored. Other preceding characters will be regarded as a fault, and trap 4 will be entered with reason 10.

A number is terminated by any character not allowed in the number, with the exception that a single space will not terminate the number: it will be ignored. Two consecutive spaces act as a terminator, as do horizontal tabulation and newline, except that spaces, horizontal tabulations and newlines between an exponent symbol (E or e) and the exponent are ignored. Whatever the terminator is, it is read in by the procedure, i.e., it is not left as the next character on the input stream.

If the number read in consists solely of a signed integer and is small enough (i.e., 32 bits) to be contained in register X, then it will be left there and PWA set to 1. Otherwise, PWA = 0 and nothing of significance is left in X. The number read will always be in register A in floating point format.

READBOOLEAN () I

RDBOOL

This searches from the current input position for one of 'TRUE', 'FALSE', 'true', 'false'. The prime symbol (') must appear in the input at each end of the letters. All symbols between the current input position and one of the above are read and ignored. On return from this procedure, the symbol after the second prime of the recognised logical value will be the next input character. If 'TRUE' or 'true' is recognised X is set to 1. If 'FALSE' or 'false' is recognised, X is set to zero. If neither is recognised, the procedure will read in all of the input stream until the end is reached.

PRINT (N, I, J)

PRINT

P2 and P3 as follows:-

(a) $P2 \neq 0, P3 \neq 0$

Fixed point format with P2 places to the left of the decimal point and P3 to the right of it. The number is rounded. It is preceded by a space, if positive, or a negative sign, if negative, and followed by 2 spaces.

(b) $P2 \neq 0, P3 = 0$

The number is printed as a P2-digit integer, rounded, and with no decimal point. It is preceded by a space, if positive, or a negative sign, if negative, and followed by 2 spaces.

(c) $P2 = 0, P3 \neq 0$

The number is printed in floating-point format in the range $1.0 \leq |P1| < 10.0$, with P3 decimal places, rounded and followed by an exponent sign (E), an exponent and 2 spaces. The number is preceded by a space, if positive, or a negative sign, if negative. The exponent is produced by OUT.I and comprises 5 characters, one of which is a space, if positive, or a negative sign, if negative.

(d) $P2 = 0, P3 = 0$

The number is printed in floating-point format with one digit, rounded and followed by a decimal point, an exponent sign (E), an exponent and 2 spaces. The number is preceded by a space, if positive, or a negative sign, if negative. The exponent format is as in (c) above.

The integers, integer parts and exponents printed by this procedure have their leading zeros suppressed. The relevant sign character is printed immediately before the first digit printed. Exponents are printed right-justified in the 4-digit positions allotted. Integers and integer parts are printed right-justified in the field allotted if P2 is large enough, otherwise they are printed left-justified and the field is extended to the right by as many digits as are necessary to complete the integer or integer part.

PRINT (R, I, I) (CONTD)

4.24

The total number of character positions used:-

- (a) $P2 \neq 0, P3 \neq 0 : P2 + P3 + 4$
- (b) $P2 \neq 0, P3 = 0 : P2 + 3$
- (c) $P2 = 0, P3 \neq 0 : P3 + 11$
- (d) $P2 = 0, P3 = 0 : 11$

P1	The number to be printed
P2	Determines the format (see above)
P3	Determines the format (see above)

26.9.73

OUTPUT

OUTPUT (R)

This procedure outputs the value of P1 in standard floating-point format. This comprises a space or negative sign, mantissa, exponent sign (D), signed exponent, semicolon and newline.

The mantissa is in the range $1.0 \leq |mantissa| < 10.0$ and comprises

an integer digit, decimal point and 15 decimal digits. The exponent occupies

5 character positions including a sign (space or negative sign) which appears

immediately before the first digit printed. It a leading zeros are suppressed.

P1 The number to be printed

26.9.73

If P1 is zero, the procedure outputs 'FALSE' (including both primes), followed by one space. If P1 is anything else it outputs 'TRUE' (including both primes) followed by 2 spaces.

P1 The logical value to be output

SPACES

SPACE (1)

This is the procedure SPACES under a different name.

P1 The number of spaces to be output.

26.9.73

NEWLINE

NEWLINE (1)

4.28

This is the procedure NEWLINES under a different name.

P1 The number of newlines to be output.

26.9.73

This uses OUT.SYM to output the formatted character.

PTNRW

PAPERTRW ()

4.29

26.9.73

4.30

RUNOUT ()

This calls OUT.SYM with a parameter of zero 60 times, thus
outputting 60 null characters, which will produce 6 inches of blank
tape if the output is sent to a tape punch.

This outputs the characters of the string described by P1 on the currently selected output stream. The string may contain any number of sets of layout-editing commands 'one-level down', i.e., the commands in each set are enclosed in special quotes.

The commands follow the rules:-

```
<set of commands> ::= <open quote><commands><close quote>
<commands> ::= <command>|<command><commands>
<command> ::= <N> P|<N> C|<N> S
<N> ::= <unsigned integer>|<null>
```

where P indicates paperthrow

C indicates newline

S indicates space

Each command is obeyed by writetext as follows:-

The unsigned integer is formed from the digits comprising it (<null> implies 1) and the specified layout-editing character is output as many times as given by the unsigned integer. If the character is C, then NEWLINES is called, otherwise CUT.SYM is used.

The code for <open quote> is 129 (i.e., %81).

The code for <close quote> is 130 (i.e., %82).

The ALGOL60 compiler will convert the various alternative symbols

it recognises as a string quote (e.g., '(', ')', etc.) into the code

for the relevant special quote.

P1 A descriptor to the string to be output

This procedure copies consecutive characters from the currently-selected input stream to the currently-selected output stream until the string described by P1 is found. It is not copied, and the characters comprising it are lost. It may be of any length less than the space remaining in the output stream's buffer at the time of matching any input character to its first character, except zero, and may contain any characters. For line-by-line output, when a character from the input stream matches the first character of the string, BREAK.CUTPUT is called in order that the output buffer is empty before COPYTEXT tries to match successive input characters with the rest of the string.

P1 A descriptor to the string to be matched with the input stream

26.9.73

This is the procedure IN.SYM under a different name.

READCH () I

IN.SYM

4.33

26.9.73

This is the procedure NEXT.SYM under a different name.

NXT.SYM

NEXTCH () I

4.34

26.9.73

This stacks X and B, calls INSYM to obtain the next input character and move the input pointer, then unstacks X and B.

SKIPCH

SKIPCH ()

4.35

26.9.73

OUT.SYM

4.36

PRINTCH (1)

This is the procedure OUT.SYM under a different name.

P1 The symbol to be output

This returns in X the ISO code of the single character in the

string described by P1.

Note that if the character is any of the string symbols for

space (i.e., \$, %, ?) then this procedure returns the code for space.

Certain 2-character strings are allowed as follows:-

EL or el	yields code for newline
EP or ep	yields code for paperthrow
SS or ss	yields code for \$

P1	Descriptor to the string containing the described characters
----	--

LIST.FILE (II, II)

This sends the file as a segment to the device specified to be listed.

P1 the filename

P2 the device/process name

OUT.DIR ()

This lists the users directory on the currently selected output stream. The procedure will use a segment as working space.

EDIT (I,I,I,I)

This procedure will edit text from an input stream to an output stream, using commands on another input stream. The 1900 implementation will destroy the contents of the X registers.

Parameters:-

P1	The input stream to be edited
P2	The output stream for the edited text
P3	The input stream containing edit commands
P4	The output stream for monitoring

Both P1 and P2 must be buffered streams.

Seven commands are recognised. The commands consist of a single letter followed by the parameters if any. No command delimiter is required except after an INSERT operation. Except during text insertion, newlines, spaces and tabs in the command stream are ignored.

The seven commands are:-

- C <N>.<N> Copy text from input to output up to (not including) the position specified (page,line, the 'page.' can be omitted if it is the current page).
- S <N>.<N> Skip input text up to the position specified.
- I <char> Insert text from the command stream into the output stream up to the first line beginning with <char>, then skip the <char> and continue reading commands.
- P <char> Insert a formfeed and continue as for I.
- L Causes the editor to remember all pages which are altered, and list them at the end of the edit.
- E Copy to end of the file and return.
- F Finish editing without copying rest of file.

In the case of both E and F, on return the original input/output streams will be selected.

Example

```
***A JOB GRF,PRG,GRFEDI
**SET.FILE.IO(1,1,'AUTO')
**EDIT(1,1,0,0)
C43.10
S11
I7
IF SB - SF > 0, -> SB.IN.SB
**STOP (0)
***Z
```

File Disc Procedures

This section gives the specification of the library procedures for the handling of discs dedicated for files.

The file disc commands will only be executed if the name of the disc specified in the command corresponds to the name of the disc held in the file directory on the disc.

SET.DISC.INPUT (I, II, II)

This procedure causes the specified disc file to be used as input (CF. SET.FILE.INPUT).

P1 is the input stream number

P2 is the disc name

P3 is the file name

SET.DISC.OUTPUT (I, II, II, I)

This procedure specifies that the output stream is to become a

file on the disc when a BREAK.OUTPUT is done on this stream. (CF. SET.FILE.OUTPUT).

P1 is the output stream number

P2 is the disc name

P3 is the file name

P4 is the cost limit

SET.DISC.IO (I, I, II, I)

This procedure causes the file to be used as input, and sets up an output stream to the disc file. (CF. SET.FILE.IO).

P1 is the input stream number
P2 is the output stream number
P3 is the disc name
P4 is the file name
P5 is the cost limit

OPEN.DISC (II, II, I, I)

This procedure creates the specified segment, and copies the disc file into that segment.

P1 is the disc name
P2 is the file name
P3 is the segment number into which the file is to be opened
P4 is the access required to the segment

FILE.DISC (II, II, I, I)

This procedure defines the specified segment to be a file on the disc. If the file already exists on the disc, then the old copy is updated.

P1 is the disc name
P2 is the file name
P3 is the segment number of the segment to be defined as a file
P4 is the access permitted to the file

DEFINE.DISC (II, II)

This procedure causes a compiled program to be defined as a file on the disc, (see DEFINE).

P1 is the disc name
P2 is the file name

DELETE.DISC (II, II)

This procedure deletes the file from the disc file directory, and releases the space on the disc.

P1 is the disc name
P2 is the file name

OUT.DISC.DIR (II)

This procedure lists the file directory of the specified disc.

P1 is the disc name

LIST.DISC (II, II, II)

This procedure sends the disc file as a message to the specified process.

P1 is the disc name
P2 is the file name
P3 is the name of the process which is to

receive the file as a message

CALL.DISC (II, II, I, I, I, I, I, I, I, I)

This procedure calls a program which has been defined as a

disc file.

P1 is the disc name

P2 is the file name

P3 - P10 are parameters as for 'CALL'.

This section gives the specification of the Library Procedures for using the System Performance Monitor (hereafter termed the S.P.M.) under process control.

1. RESERVE.SPM()
2. RELEASE.SPM()
3. CONTROL.SPM(I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32)
4. PROD.SPM(I32)
5. EVENT.SPM(I32)
6. CLEAR.TITLE.SPM()
7. TITLE.SPM(I32,I32,I32,s)
8. DISPLAY.SPM(I32)
9. OUTPUT.SPM(I32,I32,I32)
10. READ.SPM(I32)
11. WRITE.SPM(I32)
12. INT.SPM()
13. STOP.SPM()
14. SPM.FW(I32)

The SPM enables data to be collected about hardware and software activities in the MUS. This data is in the form of histograms and sets of counters.

A visual display is a part of the SPM which displays the histogram, the contents of the counters and titling information. The real store of the SPM consists of:-

1. 256 x 16 bit store which is used to contain information in the form of a histogram for monitoring activities, alternatively it may be used as 256 16 bit counters for monitoring independent activities.
11. 16 x 32 bit counters which allow monitoring of up to 16 independent hardware activities by attaching probes to the CPU of MUS. These counters may be reconfigured manually to provide 8 x 64 bit counters.
111. 256 character store which enables titling information to be displayed on the visual display.

Since the store of the SPM is of limited dimensions, a segment of the process using the SPM is required to retain the information when the store of the SPM becomes full.

A histogram conveys the relative distribution of two events A and B, where A is the more frequent of the events. The events A and B are functions of the two pulses PA and PB received by two probes attached to the CPU. A prescaling factor exists so that every second, fourth, eighth or sixteenth event of A updates the histogram.

The SPM 256 x 16 bit store may be used in four ways:-

1) Interval Histogram Mode

This is when event B causes the store location addressed by the number of occurrences of event A (scaled) since the previous event B to be incremented.

2) Dwell Histogram Mode

This is when the store records the number of occurrences of event A (scaled) between 256 successive events of B.

3) Increment Decrement Histogram Mode

This is when event A causes the store addressing register to be incremented, and the newly addressed store location to be incremented. Event B causes the store addressing register to be decremented.

4) Direct Addressing Mode

This is normally used for software monitoring, the store is used as 256 counters. SPM instructions in the executed code cause the store to be updated. The least significant eight bits of the operand of an SPM instruction addresses a store location, which is incremented if there is equality between the activity identifier in the operand and the activity being monitored.

1. RESERVE.SPM()

This procedure allows a process to reserve the System Performance Monitor. The other SPM Library procedures will set PWA negative and have no further action unless the SPM is first reserved.

This procedure releases the SPM from the calling process and makes it available to others.

2. RELEASE.SPM()

3. CONTROL.SPM(I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32,I32)

This procedure defines how the SPM will be used by this process. The program should contain monitoring identifier points (see PROD.SPM), the parameter P1 defines at which identifier points monitoring starts, and parameter P2 defines at which identifier points monitoring stops.

Parameters P3, P4, P5, P6, P7 define how the SPM is to be initialised.

While monitoring a System Performance Monitor interrupt may occur due to one of the following reasons:-

- a) In Interval Histogram or in Direct Addressing Mode when a store location overflows after being incremented.
- b) In Dwell Histogram Mode when all 256 locations have been used, or a store location overflows (this may be inhibited).
- c) In Increment Decrement Histogram Mode when a store location overflows or the store location addressed is either -1 or 256.

Parameters P8, P9, P10, P11, P12 specify the action to be taken after each such interrupt. When the required number of interrupts (P12) have occurred, monitoring either stops or Trap Procedure 6 is entered with a reason of 1.

In Interval Histogram and Direct Addressing Mode, for each interrupt the contents of the 1024 word SPM stored is dumped in the segment defined by P11, however the first 256 words of the SPM store are added to the contents of the first 256 locations of this dump area, this forms an accumulative dump, thus enabling correlated information to be gathered over a longer period of time. After the required number of interrupts (P12) have occurred or the procedure STOP.SPM is called the information collected will be scaled accordingly and displayed on the screen. A hardcopy of the collected information will be produced if requested (P8) on the output stream defined by P9.

In Dwell Histogram and Increment Decrement Mode the first SPM interrupt will cause the contents of the 1024 word SPM store to be dumped in the segment defined by P11. Any further interrupts will cause the 1024 word SPM store to be dumped into consecutive 1024 word slots of this segment. After the required number of interrupts (P12) or the procedure STOP.SPM is called a hardcopy of the collected information will be produced if requested (P8) on the output stream defined by P9, also the information obtained from the last SPM store dump will be displayed.

To produce Dwell and Interval Histograms of two software initiated events, the parameters P13 and P14 specify the identifiers for Event A and Event B respectively. Whenever the procedure EVENT.SPM is called, the parameter of EVENT.SPM is compared with these identifiers, if there is equality a software initiated event is caused and the histogram will be updated accordingly.

However, to achieve this the SPM must before monitoring be set up manually such that:-

PA = SPM instruction
PB = The least significant bit of the operand and

Parameter P6 must also be set to 3 (i.e., Event B = PA.PB).

P1 Start Identifier - whenever the procedure PROD.SPM is called with this identifier then MS02 in MS will be set.

P2 Stop Identifier - whenever the procedure PROD.SPM is called with this identifier then MS02 in MS will be reset.

P3 Defines the action of the fast counter.

Off	0
Reset and On	1
Reset and On and Override MS02	2
On	3

P4 Defines the mode in which the information is to be collected.

Off	'OFF'
Interval Histogram	'IH'
Interval Histogram and Override MS02	'IHO'
Dwell Histogram	'DH'
Dwell Histogram and Override MS02	'DHO'
Dwell Histogram with Wraparound, i.e., it to zero and continue.	'DWH'
Dwell Histogram with Wraparound and Override MS02	'DHCW'
Increment Decrement Histogram	'IDH'
Increment Decrement Histogram and Override MS02	'IDHO'
Direct Addressing	'DAD'
Direct Addressing and Override MS02	'DADO'

P5 Defines the prescaling factor for event A. This takes the value 1, 2, 4, 8, 16 and causes every event of A or every second, fourth, eighth, sixteenth event of A to update the Histogram. This prescaling factor is used only in Interval and Dwell Histogram Mode.

P6 Defines event B as a function of the two pulses PA and PB received from the CPU of MUG. This is only used in Interval and Dwell Histogram Mode.

0	event A = PA	event B = PB
1	event A = PA	event B = $\overline{\text{PA}} \cdot \text{PB}$
2	event A = PA	event B = $\text{PA} \cdot \overline{\text{PB}}$
3	event A = PA	event B = $\overline{\text{PA}} \cdot \overline{\text{PB}}$

P7 Defines the two bit activity identifier used in Direct Addressing Mode. If the activity specified in the operand of an executed SPM instruction is equal to this parameter, the SPM store location addressed by the operand is incremented.

P8 Defines the hardcopy requirements of the collected information. Bit 31 = 1 Output for 120 character position lineprinter

P9 Defines the output stream number to be used for any hardcopies produced.

P10 Defines the action to be taken after P12 interrupts have occurred.

0	Stop monitoring
1	Enter Trap Procedure 6 with Reason 1.

P11 Defines a segment in the virtual store of the calling process, which will be used to retain the information from the SPM store after SPM store full interrupts. The first 1024 words of this segment is used as a buffer for fitting information.

If P11 is negative then an unused segment will be allocated.

P12 Defines the number of interrupts to be allowed before the action defined by P10 is taken.

P13 Defines the Event A identifier (see EVENT.SPM).

P14 Defines the Event B identifier.

4. PROD.SPM(I32)

This procedure checks the monitoring identifier (P1) with the start/stop identifiers specified in the procedure CONTROL.SPM, if there is equality MS02 in MS is set or reset accordingly.

P1

Monitor Identifier

5. EVENT.SPM(I32)

This procedure compares the parameter P1 with the Event A and Event B identifiers as specified by P13, P14 of CONTROL.SPM respectively. If there is equality then for:-

Event A an SPM instruction with an operand of 0 is obeyed
Event B an SPM instruction with an operand of 1 is obeyed

This procedure has no effect unless the SPM is being used in either Dwell or Interval mode.

P1

Event Identifier

6. CLEAR.TITLE.SPM()

This procedure clears the SPM character titling buffer.

7. TITLE.SPM(I32,I32,S)

This procedure adds the specified titling information to the SPM titling buffer. For titling purposes the screen is considered as 34 lines of 74 characters. P1 and P2 specify the position on the screen at which titling is to commence.

Wherever the SPM is put into display mode, the titling strings are displayed.

P1	Defines line number where 0<P1<35
P2	Defines line position where 0<P2<75
P3	Defines the string to be displayed

8. DISPLAY.SPM(132)

This procedure stops all monitoring and then displays on the SPM visual display the histogram, fast counters and titling information either directly from the SPM store or from the area defined by P1. If the histogram store is reloaded the information will be scaled if necessary and the scale factor displayed.

P1 Defines the source of the information which

will be displayed.

0 - SPM store

Or the address in 32 bit words of the area from which the SPM store is to be reloaded before initiating display.

9. OUTPUT.SPM(I32,I32,I32)

This procedure outputs on the output stream P2 the histograms, fast counters and titling information from a previously dumped copy of the SPM store in the format as specified by P1.

P1	Defines the format of the outputs required
Bit 31 = 1 -	Output for 120 character position lineprinter
P2	Defines the output stream number
P3	Defines the address in 32 bit words of the
	SPM store dump

10. READ.SPM(132)

This procedure stops all monitoring and then dumps the contents of the 1024 word SPM store in the area defined by P1.

P1 Defines the address in 32 bit words at

which the dump is to commence.

11. WRITE.SPM(132)

This procedure stops all monitoring and then reloads the 1024

word SPM store from the area defined by P1.

P1 Defines the address in 32 bit words from which

the loading of the SPM store will start.

5.8.74/2

12. INT.SPM(>
This procedure has the effect of a System Performance Monitor
store full interrupt, thus enabling histograms etc., to be observed at
a particular point of interest within a process.

4.57

13. STOP.SPM()

This procedure first stops all monitoring and then performs the end monitoring action as defined in CONTROL.SPM.

14. SPM.FN(132)
This procedure issues an SPM instruction with P1 as the operand.
P1
Operand of SPM instruction.

Procedures For Accessing MVS Units

MASS

EX.DUMP (I, I)

This procedure gives a hexadecimal dump of mass store between the address specified.

P1 is the starting address (32 bit word address [No unit bits])
P2 is the finishing address

DISC

The following procedures are provided for transferring files

between the 1905E and the MVS disc.

The service program (test 31) should be called into MVS store

and subtest 4 entered before using these procedures.

EX.WRITE (II, I, I)

This procedure writes a file to the specified disc address. If

all of the file is to be transferred the size parameter should be set to

-1, otherwise the number of 32 bit words specified will be written to disc.

The disc address is in the format specified in the MVS Basic Programming

Manual.

P1 is the file name
P2 is the disc name
P3 is the size (in 32 bit words)

EX.READ (II, I, I)

This procedure reads the specified number of 32 bit words from

the MVS disc, and creates/updates a file.

P1 is the file name
P2 is the disc address
P3 is the size (in 32 bit words)

COPY.TO.CHART (I,I,S)

This copies flowchart descriptions in internal Flocoder form from stream P1 to stream P2 until the chart whose title is given by parameter P3 is found.

The user is faulted if:-

- a) the input stream is not of Flocoder type
- b) the requested chart occurs on the input file before the current position
- c) the output directory becomes full.

COPY.CHARTS (I,I,I)

This is similar to COPY.TO.CHART except that P3 gives the number of charts to be copied. If P3 is too large, then all the charts remaining on the input stream are copied.

The user is faulted if:-

- a) the input stream is not of Flocoder type
- b) P3 is not positive
- c) the output directory becomes full.

SKIP.TO.CHART (I,S)

This positions the input directory pointer of stream P1 just before the chart whose title is given by P2.

The user is faulted if:-

- a) the stream is not of Flocoder type
- b) the chart is not on the file.

SKIP.CHARTS (I,I)

This is similar to SKIP.TO.CHART except that P2 gives the number of charts to be skipped. P2 may be positive or negative. If P2 is too large in magnitude the pointers will be positioned after the last chart (if positive) or before the 1st chart (if negative).