SUMP

Shrivenham Universal Management Program

User Manual

SHRIVENHAM **JUNE 1974**

CONTENTS

1.0 The SUMP Operating System

This chapter gives the basic reasons for the existence of an operating system and could safely be omitted by users familiar with other 1900 series operating systems.

1.1 Executive only mode

A 1900 series machine is normally provided with an executive program written by ICL to allow programs to be controlled from a central console. On all but the smaller machines it will be capable of multiprogramming, that is of running several programs together. At any time only one program can be in an active state, i.e. using the machine's processing unit, other programs are suspended for reasons such as :-

i) They are awaiting the completion of a peripheral transfer, e.g. reading a card or writing to disc (a DA or Direct Access device),
ii) They are suspended by the operator,
iii) They would like to run, but a program of higher priority is running,
iv) They cannot proceed because of a resource clash.

Whenever an interrupt occurs (e.g. completion of a peripheral transfer or real time clock tick) or when a running program changes its state, executive selects from its list of programs the one with the highest priority it may now run.

With a careful choice of priorities, i.e. those programs with a high ratio of peripheral usage to processor usage having a high priority, this system provides an environment in which a few programs can be run efficiently.

1.2 Operating Systems

The reason why you are reading this manual is that the above method is not sufficiently successful in practise to ensure a situation in which maximum use is made of the machine – for the user this means a slower turnround of his programs. Let us consider some of the reasons why an operating system is introduced :-

i) The process of finding and running the required programs is left to a manual operator. This can cause the machine to be idle while the request is typed, it may be typed incorrectly and if the program can halt in several ways it will be necessary to provide the operator with a long list of actions to take in various situations. This rapidly imposes a limit on the complexity of processes to be carried out in a job.

ii) Peripherals such as card readers are expensive and the number of programs running together which each use a card reader is limited by the number of card readers available. It is also possible to generate conditions of 'deadly embrace' when requests for peripherals clash.

iii) Many programs have at some point during their run a sudden burst of activity on a basic peripheral (CR, CP, LP, TR, TP or GP). During this period virtually no use is made of the processor as the time scale on which the peripheral operates is so much slower. An operating system can divert this burst of activity onto disc, writing output to the actual peripheral at an orderly speed later on and reading any input documents to disc before the program starts.

iv) Two similar processes, e.g. compiling, may clash on their requirements for named disc files and there is not enough room on disc to set up multiple copies of permanent files for every situation. In other words the operator has to be told of these requirements (which are not always known to the user) and has an impossible task of producing a suitable mix of jobs.

v) By the time a large program has been developed, some of the source cards will have been through the card reader many times and will be liable to cause a card wreck – frustrating for both the operator and user. Packs of cards also spread out in a random order if dropped! It is much safer to keep such information on disc, but an operating system is needed so that the user program can be fooled into believing that the information still looks like a pack of cards. As there is insufficient room on disc to keep all this information online the operating system has to dump files not needed by current jobs to magnetic tape. This also ensures that the user has at least one secure copy of all the information.

vi) In an environment where users control the priorities of their programs it is difficult to impose standards on a wide variety of users to ensure that the simple algorithm in §1.1 is successful.

vii) A limitation (also applicable to George 2 and 3) that has particularly bothered us at R.M.C.S. is that when a job is still running at the end of a shift it can only be abandoned, a little embarrassing if it has been running for several hours.

viii) It is difficult to report to the user the complete history (monitor) of the job run. At the central console the amount of information given is limited to that directly relevant to the operator and is in serial order mixed up with messages from other jobs.

These 8 points justify the introduction of a layer of software (i.e. an operating system) between executive and the user. From the user's point of view he now has to learn the format in which he presents information to the system, but it is pointed out that this replaces a set of instructions to a human operator. Because of its different application, the job control language does not greatly resemble your programming language, it is hoped however that the stored macros described in §3, which carry out the most usual tasks, will remove most of the complexities.

2.0 Running Jobs under SUMP

The format of job input is described and the restrictions placed by SUMP on various jobs.

2.1 Inputting to Filestore

The Filestore (see §4) is used to hold the various groups of information that you need to have stored by SUMP. A given user is normally allowed access only to files under his username. Typically a file could hold the source of a program under development or data for a program. To input a new file the set of records (on cards or paper tape) is presented to SUMP, preceded by an INP card and ending with a terminator card which is not itself included in the new file.

The INP card is of the form :-

```
INP user,title,Txxxx
```

where

i) 'user' is the username as known to SUMP.

ii) 'title' is the name of the new file being introduced.
Both 'user' and 'title' are a set of characters satisfying the restrictions of an exofile name, i.e. up to 12 characters long, consisting of letters, digits, hyphens and spaces, the first character being a letter. Later references to the file appear as :title.

iii) 'xxx' are any 4 characters distinct from the first 4 characters of records in the file. If Txxxx and the preceding comma are omitted a default of **** is assumed.

The terminator card consists of a record having xxxx as above for the first 4 characters.

E.g.

```
INP HUNTER,PROG-SOURCE,T++++
data records
++++
INP SUGDEN,DATA MK 3
data records
****
```

Files input by the INP method are permanent and will be retained by the system.

2.2 Running a Job

A job starts with a record of the form :-

```
JOB user,title,Txxxx
```

where

i) 'user' is the username as known to SUMP, which determines as a result the limits to place on the job, see §2.3.

ii) 'title' as in §2.1, but it is used simply to identify this job from other user runs.

iii) 'xxxx' being the terminator as in §2.1

The terminator shows the end of the job and encloses

i) Any basic command instructions – see §5.
ii) Macro calls – see §3 and §6.
iii) Temporary documents

A temporary document is a set of data required to be held by SUMP only for the duration of that job. A temporary file is set up in filestore and automatically erased at the end of the job. Such a document starts :-

```
DOC title,Txxxx
```

and ends with the terminator xxxx (or the default value of **** if no terminator has been given). The title gives the name of the temporary file set up, there is no user specified as the entire document is embedded in a job giving SUMP the username.

E.g. a complete job could consist of :-

```
JOB HUNTER,TEST,T++++
FORTRAN PROG-SOURCE
SA :TESTDUMP
RUN
DOC DATA
    data records
****
++++
```

The above uses the file :PROG-SOURCE previously input as the FORTRAN source; the macro FORTRAN (see §3) to produce the binary form of the program; the basic command SA (see §5) to set up a new filestore file to keep a copy of the binary; the macro RUN (see §3) to run the program using data from DATA.

To run the same program on different data, while avoiding the expensive and time consuming process of retranslating the program source, we could use :-

```
JOB HUNTER,RE-TEST,T++++
LO :TESTDUMP
RUN
DOC DATA
    new data records
****
++++
```

2.3 Job limits

SUMP is a batch operating system designed to load the machine with an efficient mix of jobs in, say, three streams referred to as A, B and C. A given user is associated with a given stream and he should choose the appropriate stream referring to the limits below. The A stream consists of 'student' type jobs having low values for the limits on time and volume of output. The B stream is for development work and the C stream for production runs. The B stream allows a user to get a rapid turnround while a program is being developed, even if it has large core requirements.

The values for the limits are controlled by the system manager and can be changed in light of experience, SUMP (3A) is issued with limits :-

	A	B	C
Default Time limit (seconds)	75	300	3000
Maximum Time	105	600	6000
Default Volume (records)	300	1500	2500
Maximum Volume	600	3000	5000

Each user may have up to 30 files in filestore, the maximum size of a filestore file being 800 blocks in SUMP(3A).

Note that the volume limit refers to the number of records actually produced on the line printer, card punch etc., not to the number of records passed to filestore, although a program exceeding the limit will halt illegal Z.

2.4 Job output

The output from a SUMP job consists of those listings generated by the user during the run of the job, followed by a monitor file. The monitor file will probably appear somewhat overpowering at first sight, but it describes the run of the job from SUMP's point of view. Needless to say, if you have not had the output you expected from the job, this should be the first thing to look at for an explanation. The monitor file consists of three parts :-

1) The commands to be obeyed, including those generated from macro expansions, in the order as given by the user. SUMP adds a physical line number so that it can refer to the particular line if it causes an error when it is carried out. A macro reference is followed by its expansion, the expansion lines being indented. Note that macros may include an SU command to suppress the listing, in which case there will be a jump in the line numbering. An error at this point is usually the result of an incorrect use of a macro, e.g. when an actual parameter is missing; the line 'so far' is printed and an error message given.

E.g.

```
FINDGO
    FI #
        ↑
    MISSING PARAMETER IN FI #%A,:PROGRAM FRED
```

The trouble here resulted when an attempt was made to substitute characters for %A in the macro definition line, FINDGO being called with no actual parameters.

Such erroneous lines are ignored during the run of the job.

An extra line with the EJ (end job) command is automatically added as a final command.

2) The second section shows the progress of the job. The time, line number and command obeyed are given as they occur, interspersed with program events such as halts, displays and change of core size. As it is only at this stage that a check is carried out to establish that the command makes sense, an error message (marked by **) may appear on the next line. Comment lines (marked by >>) also appear.

E.g.

```
12.03.34              >> STARTED
12.03.34      LINE  1    LO :FREDUMP(7)
12.03.36      LINE  2    ER :FREDUMP(6)
12.03.36      LINE  2 ** NO SUCH FILE
12.03.36      LINE  3    ER #FRED
12.03.36      LINE  3 ** PARAMETER 1 WAS PROGRAM NOT :FSTORE
12.03.36      LINE  4    EN 0
12.03.36     #FRED    HAL-TR
```

If a program obeys an illegal instruction, details are given which may help to identify the cause, although these details require a knowledge of 1900 machine code :-

i) If the instruction concerns a peripheral device, then the device and unit in question and the control area are printed. If the device has not been allotted, this fact is also output.

ii) The instructions in the neighbourhood of the illegality, 15 before and 5 after the offending instruction.

iii) The accumulators and word 8 of the offending program.

The core prints are given in the standard form as described under the PR command.

Special treatment is given to the IF command in the event of the condition being true, when the conditional command is reprinted on a new line.

E.g.

```
12.03.36      LINE 26    IF DEL<>'FI' GO 1OUCH
12.03.36      LINE 26       GO 1OUCH
12.03.36      LINE 38    1OUCH LF :OUTPUT,*LP
```

3) Lastly there is a section which shows the state of the user's portion of filestore at the end of the job and certain job statistics.

In the absence of an LD command, only a summary of filestore is given:-

E.g.

```
RESULTANT FILESTORE
            8 FILES, TOTAL SIZE 364 BLOCKS
```

If the LD (List Directory) command has appeared, details are given for each file. These details include Name, Generation No., Size in Blocks, Age in days, Days since last access, No of accesses, Staleness and Type. The staleness of a file is a changing value depending on the size and age of the file. This value is used by SUMP in deciding whether the file can be kept online. The type of a file refers to whether the file is being used for a basic peripheral, or for a Direct Access (DA) or Magnetic Tape (MT) use.

All the LD information is printed between the line `RESULTANT FILESTORE' and the summary line.

Finally, a summary of the resources used by the job as a whole:-

i) The mill time used by the job, i.e. The central processor usage caused by the job, excluding such items as the listing of files which count as operating system activities.

ii) The total number of records output by the job, i.e. the sum of the lines printed on *LP and the cards punched etc.

iii) The number of records generated by the programs run in the job.

3.0 Standard Macros

For the user who wishes to run certain standard jobs (with small variations provided by actual parameters) the definitions of a set of macros follows.

For a full definition of a macro see §6, for the present chapter it will suffice to think of the macro as a set of stored commands. These macros should however be regarded as no more than an example in the use of SUMP macros and an easy method of starting to use SUMP, they will be changed if found unsuitable.

3.1 Standard Parameters

After the macroname call a set of optional parameters may be added, separated by commas. If all parameters are not given, the position of later parameters has to be shown by giving all the significant commas. Trailing commas may be omitted. E.g. If the macro SLANG is to be called with the third parameter having a value of 6, all other parameters being default we write:-

```
SLANG ,,6
```

Standard default values are given for at least first three parameters in the macros. Following the latter notation, the first parameter is referred to as %A, the second %B and etc., a default is enclosed in < >.

Six standard macros are given to compile a program in the languages:

FORTRAN, COBOL, ALGOL68, PLAN, CORAL and ALGOL60

these being the names of the respective macros. Note that after one of these macros the core image is at the point when the user's program is ready to run.

The first three parameters are common to each definition and give:-

%A <ANON> = Program identifier, a four character name (starting alphabetic and thereafter alphanumeric).

E.g. ABCD, X101, PP25

This will become the name of the program when preceded by # and be used as part of the title of various files.

%B <SOURCE> = the name of the document or the filestore entry from which the compiler is to read records. Note that it is better to INPut to filestore and edit if necessary for large source programs.

%C<0> = a steering value to provide certain options. So that any combination of options can be constructed the value is given being the sum of those required from

1 = user job description, rather than the default.
2 = suppress the compiler listing if it was successful.
4 = preserve the semicompiled output in the file called :%A<ANON>SEMICOMP(0)
8 = compile only

E.g. a value of 5 is given if the user gives his own job description and wishes to keep the semicompiled.

Any further parameters are peculiar to the language as given in the next sections. After reading about your particular language go to section 3.8.

3.2 FORTRAN

Further parameters are :-

%D<2> = trace level in the default job description. Make sure that this is 0 in the compilation of the binary program used for production runs.

%E<XFAT> = compiler name. XFAT is an extended FORTRAN compiler giving good diagnostics and reasonable code. XFEW is an optimising compiler giving the best quality code for production runs. XFAE is faster but gives less efficient code and numerical diagnostic messages. If in doubt, use XFAT for development and short runs, long/repeated production runs being done with a program compiled by XFEW.

The job description for FORTRAN compilers consists of two sections. The macro provides the first section irrespective of the value of %C :-

```
LIST(LP)
SEND TO(ED,%A<ANON>SEMICOMP)
WORK(ED,FORTRAN WORK(0))
WORK(ED,FORTRAN WORK(0))
```

The second section, which is avoided if %C is odd, consists of :-

```
PROGRAM(%A<ANON>)
INPUT 1,5=CR0
OUTPUT 2,6=LP0
COMPACT
TRACE %E<2>
END
```

3.3 COBOL

Further parameters are :-

%D<COBOL> = List option for the *LIST line. COBOL MAP would be the normal alternative.

%E<*CON> = a line in the steering file which can be replaced, e.g. by *LIB.

%F<> = an optional additional libe, e.g. *SUBROUTINES.

The compiler is always XEKB, if %C is odd the user should supply the entire steering file. The default steering file is :-

```
*IDE %A<ANON>
*COM
*COBOL CARDS (%A<ANON>SOURCE)
*OBJ EDS (%A<ANON>SEMICOMP(0))
%E<*CON>
*LIST %D<COBOL>
%F<>
****
```

Two points to be noted by George II users,

i) A **** card must be included at the end of the source file, and at the end of a user provided steering file.

ii) The Cobol program should use unit number 0 for LP and CR. If other units are used, they must be ASsigned to appropriate files before RUN is used.

3.4 ALGOL68

%D<4K> = size for object program

The first two lines required by the compiler, i.e. AL68 and the name %A<ANON>, are always provided. Ignore the use of values 1 and 4 in %C.

3.5 PLAN

Only XPLT is available at RMCS for reasons of disc economy, the use of filestore removes the need for the cosy system and all PLAN 3 facilities are implemented in PLAN 4.

%D<1> = number of segments in the source.
%E<SUBROUTINES> = the subfile name in the compiler output file.
%F<LIST> = listing level.

The input is always preceded by the steering lines

```
PROG(%A<ANON>)
OUT(%A<ANON>SEMICOMP)
WSF(%E<SUBROUTINES>)
STEER(%F<LIST>,OBJE)
```

If %C is even, automatic consolidation is caused by the further lines :-

```
NEXT(#XPCK#SCIN)
BIN
PLAN(CR,%D<1>)
```

3.6 CORAL

%D<2> = Dump exofile. Four characters xxxx given here will result in the program being dumped to an exofile PROGRAM xxxx. If no actual parameter is specified the effect is to cause a harmless error, removing the DUMP line from the job description.

%E<'TRACE'> = an extra job description line. To remove trace a null parameter " " could be given. More than one control can be placed there as a single parameter, e.g.

```
"'LIBR'(SUBGROUPSRF7,SUBROUTINES) 'NOLIST'"
```

Note the need to enclose the whole group inside double quotes because of the comma.

The following job description is always given :-

```
'LIST'
'PROGRAM' (%A<ANON>)
'SEND TO' (%A<ANON>SEMICOMP,SUBROUTINES)
```

If %C is even, there is the continuation :-

```
'DUMP ON' (PROGRAM£%D,PROGRAM£%A<ANON>)
%E<'TRACE'>
'LIBRARY' (SUBGROUPCORA,SUBROUTINES)
'LEADERS'
****
```

3.7 ALGOL60

Further parameters are :-

%D<2> = Trace level. Make sure that this is 0 in the compilation producing the program for production runs.

%E<XALT> = name of the compiler. XALE is the alternative (it is a little faster, but produces less efficient object programs).

The following job description is always given :-

```
'LIST' (LP)
'SEND TO' (ED,%A<ANON>SEMICOMP,.SUBROUTINES)
'WORK' (ED,WORK)
```

If %C is even, it continues with :-

```
'PROGRAM' (%A<ANON>)
'COMPACT DATA'
'INPUT' 0=CR0
'OUTPUT' 0=LP0
'TRACE' %E<2>
```

3.8 RUN

RMCS George II users will immediately notice a change of policy, in that each of the above macros ends at the point where consolidation is complete, i.e. when the core image is that of the user's binary program which is now ready to be entered. Under George II there was nothing better to do than to run the program. Under SUMP it is easy to use the SA command if a copy of the compiled program is required, so that the same program may be run again without recompiling by using the LO command, (e.g. §2.2).

To run the program there is a macro RUN with the parameters :-

%A<ANON> the same program name as used for the %A of the language macro, this is not critical however as it is used here only to construct the name of the LP output file.

%B<DATA> the name of the document or filestore file holding the data,if any, for the program.

%C<0 (unless Algol 68 when 1)> - entry point.

4.0 Filestore

The user should view the Filestore not as the fluid area of disc and magnetic tape that it consists of in reality, but as a region that he alone can write to, and as a region where he may store various collections of records.

The limitations placed on the area available to a user are less for intrinsic system reasons than to encourage the user to erase obsolete items. The filestore is dumped frequently (say every two hours) to tape, so that the user is safeguarded against any spectacular disasters.

The generation number of the file is part of its identification, and serves to distinguish between different versions of a file, e.g. before and after an edit. A file of generation zero is set up by default in an INPut run. Certain commands, e.g. CE and ER, insist that a generation number be specified as part of the file name; otherwise if the generation number is omitted the file of the given name with the highest generation number is chosen.

As the use of the filestore grows, it is no longer possible to keep all of it on discs online, those files first to be dumped onto magnetic tape are those which are the oldest and least frequently used (i.e. having a large 'staleness').

The four types of files are described in the next four sections.

4.1 User Permanent Files

These files are kept within the system between user runs. They may be created by one of four methods :-

i) INPutting a deck of cards using the INP directive
ii) From the output of a program, by ASsigning a line printer, card punch or tape punch to the file.
iii) By EDiting an already extant permanent file.
iv) By SAving a core image.

At the end of a user job all such files are safeguarded by the system for future runs. Permanent files are erased by means of the ER command, or by an edit with no second parameter. The system only maintains the highest generation number of each permanent file between filestore dumps.

4.2 User Temporary Files

These files exist only for the duration of the particular job. They are created either by the DOC directive or by onlining an output peripheral to a file.

4.3 System Work Files

System generated files, e.g. for the offlined input and output, are automatically removed when no longer needed.

4.4 User Direct Access Files

These are permanent files for use, e.g. by compilers, as direct access work files. They are created by use of the CE command, which specifies a size which cannot be changed during the job. The assignment command AS linking a peripheral to such a file, diverts it to filestore instead of the exofile. It is however a sensible convention to choose a filestore file of the same name as the exofile that the program believes it has opened.

5.0 Commands

The basic commands recognised by SUMP have the characteristics :-

i) They consist of a two letter mnemonic, e.g. LF.
ii) Optionally, they may be preceded by a label
iii) They are followed by parameters separated by commas, certain commands are allowed an expression rather than a single parameter,
iv) Except for the IF, only one command appears on one line.

They are of four types :-

i) Job control (J)
ii) Program control (P)
iii) Filestore control (F)
iv) Edit (E)

5.1 **The complete list of commands,** in alphabetic order. is :-

In the command definitions which follow the parameters are referred to as E1, E2 etc. if expressions are allowed; S1, S2 etc. if simple entities only are required. Bracketed parameters are optional.

5.2 Job control commands

These commands are the general purpose ones concerned with controlling various aspects of the manner in which the job is to proceed.

5.2.1 **DP** (Display)

Displays a message to the operator upon the central console. The message may be up to 40 characters in length and must not include the characters $,], ↑ or ←.

S1 Text string.

E.g.

```
DP 'PLEASE LOAD SCRATCH TAPE'
```

5.2.2 **EJ** (End Job)

Ends the current job.

There are no parameters.

E.g.

```
IF HALT<>'AH',EJ
```

5.2.3 **GO** (Go)

Jump to the point in the commands having the specified label.

S1 Label.

{In SUMP (3A) if the label is not unique, the first occurrence of the label will be used.}

E.g.

```
IF TIME>200,GO 2A
```

Error :-

More than 64 jumps being made during the run of the job.

5.2.4 **IF** (If)

Causes a subsequent command on the same line to be executed or ignored depending respectively on whether a logical expression is true or false when the IF is encountered.

E1 Logical expression.

The subsequent command is separated from the expression by a comma.

E.g.

```
IF @9 MASK 4=4 AND @8 = 0, LF :OUTPUT,*LP
```

5.2.5 **LD** (List directory)

Causes a full listing of the user's filestore entries to be given at the end of the monitor file.
There are no parameters,

E.g.

```
LD
```

5.2.6 **SK** (Skip)

Causes the system to ignore a number of lines following, if the first parameter is null.

(S1) Any or null.
S2 Numeric. The number of lines to be skipped.

This command will normally appear only in macro definitions.

E.g.

```
SK %F,1
AS *LP%F,:PRINTER%F0-P
```

5.2.7 **SP** (Set Parameter)

Sets parameter, i.e. a user variable as described in §8.1.

E1 Numeric, in the range 0 to 9. This specifies which variable is to be altered.
E2 Numeric. The new value to be given. Note that if this parameter or the previous is an expression, the value assigned and the variable concerned will be printed in the monitor file.

E.g.

```
SP 1,0
10    IF HALT<>'ST',GO 11
DP '%U %X NEEDS MORE CORE'
WT 60
SP 1,@1+1
IF @1<5,GO 10
EJ
11 ....
```

Error :-

E1 outside the required range.

5.2.8 **SU** (Suppress)

The monitor file listing at this and any lower levels of macro nesting is suppressed.
There are no parameters.

5.2.9 **TM** (Time maximum)

Sets a time limit at which SUMP will send the program illegal 'W'. If the user does not specify a limit, SUMP sets up a default limit depending on the stream in which the job is run.

E1 Numeric. Units of seconds.

If the value at which the program is sent illegal is less than the stream limit, TM can be used to reset the limit and the program re-entered (possibly to wind up in a useful fashion).

E.g.

```
TM 60
```

Error :-

Value given is larger than the stream maximum, command ignored.

5.2.10 **VM** (Volume limit)

Sets a volume limit which operates in two ways. SUMP will send a program illegal 'Z' if the core images in the job have output more records than this limit, this limit also controls the number of records that can be offlined to be finally printed on the lineprinter etc. SUMP sets a default value if this command is not used.

E1 Numeric. The total number of records.

Error :-

Value larger than the maximum for the stream.

5.2.11 **WT** (Wait)

Suspend the job, usually to allow the operator time to alter the environment, e.g. to load a tape or make more core available.

E1 Numeric. Units of realtime seconds.

Error :-

Value larger than ten minutes.

5.3 Program control commands

This group of commands has an effect on the core image.

5.3.1 **AL** (Alter)

Alters a specified word in the current core image.

E1 Address of the word to be altered. Numeric.

E2 New value. Numeric.

The user may be tempted to alter word 8 of the core image in an attempt to cause the program to resume from a different instruction. Word 8 should be regarded as “read only”, giving information as to the current state of overflow and next instruction; any use of SUMP to change the next instruction to be obeyed in the core image must use the EN command.

E.g.

```
AL 45,1234
AL [[8] MASK #17777777] MASK #7777,0
```

Errors :-

i) No core image exists
ii) The address specified is out side the core image.

5.3.2 **AS** (Assign)

Assigns a peripheral to a filestore file.

S1 Peripheral name.
S2 Filestore file.

For an input device the file must exist already. For an output device the file is created if it doesn't exist already; if it exists as a temporary file an error is flagged. If the peripheral is already attached to a file, the old file is immediately closed.

In the special case of the peripheral being *DA or *MT, the file concerned must have been previously created by CE.

E.g.

```
AS *CR0,:DATA
AS *DA1,:ANONSEMICOMP(0)
```

Errors :-

Violations of the above conditions, resulting in either the message “no such file” or “unsuitable file”.

5.3.3 **EN** (Enter)

Enter or resume at a given instruction in the current core image.

(E1) Numeric. If in the range 0 to 9, then 20 is added so that the conventional entry points are used. If no parameter is given the program is resumed at the current instruction.

If the value of E1 is outside the core image an immediate illegal 'X' can be expected.

E.g.

```
EN 0
EN
EN [8] MASK #77777 + 2
```

Error :-

No core image

5.3.4 **ET** (Entrust)

Inform SUMP that the current core image is a trusted program. The effect is to allow the 153 instruction to be used as a command issuer and limited trusted facilities such as needed by FLAIR.

There are no parameters.

If the 153 instruction is used, the X field must be 5 and X5 holds the length of the message in characters. N(M) is the message area and the message will be used by SUMP as a job command.

5.3.5 **FI** (Find)

Finds a program from a conventional program library held on an exofile or exotape. To find a program from exotape the FI command must be followed by a LO without parameters.

- S1 Program name.
- S2 Exofile name.
- (S3) Core size. If this value is greater than the size in the program request slip, it will be used instead.
- (S4) A text string. This causes an EVENT of this value to be generated at the end of the loading process. It is intended for use with program such as #XPCK, so that assigns may be carried out before the 'GO' entry causes the program to start without an EN. The instruction monitored is the 'ALLOT 0 62'.

If the core size given as the optional S3 is less than the program size, the loader will halt 'E2'. FI destroys any existing core image, except for the accumulator values and releases any *DA or *MT peripherals.

E.g.

```
FI #XFAT,PROGRAM SCIN
FI #CORA,PROGRAM ADVA,25K
IF DEL='FI',FI #XPCK,PROGRAM SCIN,,'OKAY'
```

Errors :-

- i) Program not found.
- ii) Program too large.

5.3.6 **LO** (Load)

Loads a program which has previously been SAved or INPutted, or a program on magnetic tape.

- (S1) Filestore name. If absent the previous command must have been FI to find a program from *MT.
- (S2) Optional core size (as for FI).

An overlaid program cannot be LOaded, it must be found from a disc file.

E.g.

```
LO :FREDDUMP
```

Errors :-

- i) The file does not contain the standard ICL binary card format.
- ii) Checksum errors.
- iii) Certain features, e.g. multimember program are not implemented.

5.3.7 **OF** (Offline)

Attaches a peripheral (*TP, *CP or *LP) to a magnetic tape. This facility is intended to be used when larger volumes of output are produced for printing later by a utility program. The record format and other details are given in the system manual.

S1 Peripheral
S2 Name of output tape created, retention period 28 days.

E.g.

```
OF *LP0,FRED LP0 OUT(4)
```

Errors :-

i) Peripheral neither *LP, *CP or *TP.
ii) No scratch tape available. An event 'FAIL' is generated in this case.

5.3.8 **OL** (Online)

Attaches a basic peripheral to a filestore file. If this file does not exist and the peripheral is an output device, a temporary file is created.

S1 Peripheral.
S2 Filestore name.

E.g.

```
OL *CP,:CARD OUTPUT
```

Errors :-

i) No such file exists and the peripheral is input.
ii) A permanent file already exists and the peripheral is output.
iii) A *DA file of that name already exists.

5.3.9 **PR** (Print)

Gives a core print of a portion of the current core image. Three words are printed per line, in octal, instruction and character formats. Repeated areas of core are not listed in full, but a blank line is left to show this occurrence. Instruction mnemonics (see PLAN manual) are used, but where one function code gives rise to several mnemonics a compromise, e.g. BR?, is printed.

(E1) First word to be printed. If not given, zero is assumed.
(E2) Last word to be printed. If not given, the last word of the program is assumed.

E.g.

```
PR ,200
PR [8] 'MASK' #77777
PR 1738,2001
```

Errors :-

i) Outside program. (E1).
ii) E2 greater than E1.

5.4 Filestore commands

These commands are concerned with filestore manipulation.

5.4.1 **CE** (Create)

A new fast peripheral (*DA or *MT) file is created in filestore for the user. It will be a permanent file of constant size.

S1 Filename including generation number.
S2 Size in words.
S3 Peripheral type, *DA or *MT. (N.b. *ED is the same as *DA).
(S4) Blocks per bucket, default 1 block buckets.

Such files are frequently used as work or output files by compilers.

E.g.

```
CE :WORK(2),10K,*DA,4
CE :LIB %Y(0),5K,*ED
```

{Setting up a file with 1 block buckets with the current date as part of the name}

Errors :-

i) File of that name already exists.
ii) Size greater than 100K.
iii) S3 incorrect type.
iv) S4 given, but neither 1, 2, 4 or 8.
v) No generation number given.

5.4.2 **CF** (Condense Filestore)

All temporary files, including DOCuments, are erased.

There are no parameters.

5.4.3 **ER** (Erase)

Erases a file from filestore.

S1 Filename.

If no generation number is specified, then the highest generation in use of files of that name is selected.

E.g.

```
ER :LIB 19-08-74
```

Error :-

No such file exists to be erased.

5.4.4 **JD** (Job Description)

A temporary file is created from lines following in the commands to SUMP. The file is of basic peripheral type and will normally be used as a job description for a compiler. When this command is used in a macro, any references to formal parameters will be substituted by the actual parameters.

S1 Peripheral concerned.

S2 Number of records in the file, i.e. the number of lines following the JD line which are not taken as commands, but as the lines in the file. N.B. this parameter is altered by SUMP to be an internal variable for use when the JD command is carried out.

A secondary function of the command is to assign the peripheral to the file when the JD is obeyed.

E.g.

```
JD *CR0,2
      LIST
      SEND TO (ED,%A<ANON>SEMICOMP(0))
```

5.4.5 **LF** (Listfile)

Gives a copy of the whole or part of a filestore file on a basic peripheral. If either or both of the optional E3 and E4 is given then lineprinter output is numbered and cards have a sequence number placed in columns 73-80. On completion the file is closed and if it was assigned/onlined to a peripheral it is released.

S1 Filename. The highest generation is used if no generation is specified.

S2 Output peripheral.

(E3) First record to be output, default is 1.

(E4) Last record to be output, default is the last record of the file.

E.g.

```
LF :PRINTER-2,*LP
LF :FRED SOURCE(13),*LP,1
LF :FRED SOURCE,*CP,35,97
```

Errors :-

i) S2 neither *LP, *TP or *CP.

ii) No such file exists.

iii) E4 is less than E3.

N.b. If either E3 or E4 is used, information in the record may be lost, i.e. columns 73-80 of a card or positions 113-120 on the lineprinter.

5.4.6 **SA** (Save)

Saves the current state of the core image in the form of a standard ICL card dump. The core image may be loaded at a later date from the permanent filestore file created by this command.

S1 Filename

E.g.

```
SA :FREDDUMP
```

Error :-

No core image exists.

N.b. Only the core resident part of an overlaid program is saved.

5.5 Editor commands

Details of the edit are given in section 7, the five commands concerned are given below as the editor is inbuilt and SUMP treats editor command no differently from other commands.

5.5.1 **CO** (Copy)

Records are copied from a second file into the edited file.

E1 Line number of old file after which the insert is to be placed.
E2 Line number of the second file to start copying from.
E3 Last line of the second file to be copied. A value of zero is equivalent to 'copy to end of file'.
(S4) Name of the second file. The default is to use the first file to copy into itself.

E.g.

```
CO 17,3649+379,3872+379,:ODDMENTS(2)
```

Errors :-

i) Reference outside the bounds of the file.
ii) E3 differs from zero and is less than E2.

5.5.2 **DE** (Delete)

Delete records from the edited file.

E1 First line to delete.
(E2) Last line to delete. The default value is the same line as E1.

E.g.

```
DE 1984,2001
DE 23
```

Errors :-

i) Reference outside file.
ii) E2 less than E1.

5.5.3 **ED** (Edit)

Starts a group of editing instructions, terminated by TE.

S1 Filename of the old version of the filestore file. If the generation number is not given, the highest available will be used.

(S2) Filename of new version. If given the old version will be retained, if absent the old file will be erased and the new version left in a file of one higher generation than the old version.

(S3) File containing edit commands. Default is to take the edit commands from the job control.

E.g.

```
ED :FREDSOURCE
ED :TESTDATA(1),:LONG-RUNDATA(1)
ED :SUMP SOURCE(220),,:ed 220-221
```

Errors :-

i) A previous ED in the same job has not been terminated by TE.

ii) File S1 does not exist.

N.b. An error is given to any other edit commands if they are not preceded by an ED.

5.5.4 **IN** (Insert)

Insert a given number of records which follow the IN record, into a given position in the edited file.

E1 Line number of old file after which records are inserted.

E2 Number of lines, following the command, to be inserted.

E.g.

```
IN 97,1
AMENDMENT LINE
```

Error

Not in the file.

5.5.5 **TE** (Terminate edit)

Shows the end of the group of edit commands starting with the ED. At this point the editing process is actually carried out.

There are no parameters.

If SUMP detects that there are editing commands outstanding when an EJ is met, it will carry out a TE before ending the job.

6.0 SUMP Macros

A macro is a stored collection of text which is expanded for the user when he makes a reference to the appropriate name. The expansion will consist of a series of commands and job descriptions. There is a certain amount of flexibility provided in these expansions as the user may provide actual parameters in the macro call which replace dummy references in the macro definition.

The notation used is to refer to up to 20 dummy parameters in order as %A, %B, %C etc., and if an optional default is given it is placed between afterwards.

E.g. The macro FINDGO could consist if the two lines :-

```
FI #%A,%B<PROGRAM UTIL>
EN %C<0>
```

This macro finds the program name given as the first parameter (without the #) from the file given in the second parameter. It then enters this program at the location given by the third parameter, or if the third parameter is absent, the default value of 0 is used. Note that at this stage (the first section in the job monitor) no checks are carried out as to whether the final result makes any sense, it is just a process of character substitution.

The dummy parameters from %U onwards are special system values instead of being associated with parameters in the macro call. They allow the user to introduce system values into the macro expansions.

%U User name
%V Volume limit for the stream in which the job is run
%W Mill limit for the stream
%X Job name
%Y Date in the form 07-12-75
%Z Stream in the form A, B or C

6.1 Macro expansions

Three references to the macro FINDGO (see §6.0), could result in part of the monitor file appearing thus :-

```
17   FINDGO FRED
18       FI #FRED, PROGRAM UTIL
19       EN 0
20   FINDGO JOEY, PROGRAM JUNK, 7
21       FI #JOEY, PROGRAM JUNK
22       EN 7
23   FINDGO 19,,ZERO
24       FI #19, PROGRAM UTIL
25       EN ZERO
```

The line numbers on the left hand side are introduced by SUMP, as is the indentation to show that a macro expansion has occurred. Because of the unsuitable parameters on line 23, errors will occur when lines 24 and 25 are obeyed. Note on line 23 the need for two commas to show that ZERO was to be substituted for %C rather than %B.

If a macro is called unsuitably, a printout is given of the expansion as far as was possible, followed by an error message. The line in error will not be obeyed.

E.g.

```
26   FINDGO ,2,6
         FI #
             ↑
           UNDEFINED PARAMETER IN FI #%A,%B<PROGRAM UTIL>

27       EN 6
```

A macro definition may itself contain a non-recursive reference to another macro, nesting macros up to the limit of a depth of 7 in the current version of SUMP.

A macro name consists of an alphabetic character followed by up to 11 alphanumeric characters, provided it is distinct from a basic command name of the word END.

The next section describes how the user may set up his own macros in a filestore file. To notify SUMP that a particular file holds macro definitions, the name of the file is given as the only item in a line in the job commands.

E.g.

```
:MTMACROS
```

SUMP will now, on finding a reference to a macro, search the file :MTMACROS to see if the macro is defined there. Only if the search is unsuccessful will SUMP proceed to search the system macrofile. The user can therefore set up a private redefinition of one of the system macros, as well as any special purpose macros for his individual needs.

There are certain occasions in which the substitution of actual parameters into the macro definition is complicated, i.e. when the characters used include comma, percent or space.

E.g.

```
MACRO1  is   DP '%A'
             FINDGO %B,%C<PROGRAM TEST>
```

and a straightforward use would be :-

```
28   MACRO1  NEEDS DISC 17,DBS1
29       DP 'NEEDS DISC 17'
30       FINDGO DBS1,PROGRAM TEST
31           FI #DBS1,PROGRAM TEST
32           EN 0
```

Note however the effect of an attempt to generate the message

```
'IF IT LOOPS, ABANDON'
```

by

```
33   MACRO1  IF IT LOOPS, ABANDON,DBS1
34       DP 'IF IT LOOPS'
35       FINDGO ABANDON,DBS1
             FI #ABAN
                   ↑
              COMMA EXPECTED IN FI #%A,%B<PROGRAM UTIL>

36           EN 0
```

Space characters also cause trouble, e.g. is the second actual parameter of line 35 to be interpreted as DBS1 or DBS1 followed by about 40 spaces? Following GIN (the language in which SUMP is written) conventions, any group of consecutive enclosed spaces is regarded as a single space character, trailing groups of spaces are ignored. This is however a nuisance if a filename of the form :ABCD IJKL is required. A % symbol and the character following it are taken as a reference to an actual parameter from the higher level. In any of these cases the entities may be protected by enclosing all, or the critical parts, inside double quote marks. A pair of " " is stripped from an actual parameter when it is substituted n a macro definition, a pair of adjacent "" is interpreted as ". An artificial example to make all this clear is to use MACRO1 to display

```
50% "WORKING NOW"
```

followed by a run of #FRED held on the file FRED Z(6).

```
37   MACRO1  "50% ""WORKING NOW""",FRED,""FRED     Z""(6)
38       DP '50% "WORKING NOW"'
39       FINDGO FRED,"FRED     Z"(6)
40           FI FRED,FRED     Z(6)
41           EN 0
```

6.2 Macro Files

These files, set up either by the user or the system, hold the definitions of the various macros. Such files may be edited in the normal way.

The records on the macrofile consist of

i) A list of macros defined in the file.
ii) An END record to show the end of the list.
N.b. this precludes the name END being used as a macroname.
iii) The definition of each macro, not necessarily in the same order as (i).

Each definition has for the first record the name of the macro, followed optionally by a space and a comment. It is suggested that this comment could be used to specify the parameters. The end of each macro definition is shown by END.

E.g.

```
SUMPJUNK
FRED
JOBRUN
END
SUMPJUNK IS FOR TESTING ONLY
DP %A<'EUREKA'>
EJ
END
JOBRUN FILE, ENTRY POINT<0>
LO :%A
EN %B<0>
END
FRED SWITCH, OK HALT
LO :FREDFILE
AL 30,[30]UNION(1 SLC(23-%A))
EN 0
IF NOT HALT='%B',,EJ
END
```

There is no reason why a user should not have several private macrofiles, although at any point in the job commands, it is only possible to make SUMP search one file, i.e. the last one whose filename has been given as the current user's macrofile.

7.0 Editor

To allow the user to use the filestore instead of assembling large packs of cards, a simple and flexible editor is integrated into the system. It is envisaged that the typical use would be to edit the source of a program from a previous listing. The LF command can be used to give a listing with line numbers starting from one, although it is likely that the user would have a compiler listing with line numbers in any case. If the compiler does not list from line 1 or if a job description from a system file precedes the user file it could be necessary to subtract an offset from the editor commands, as in the example below. One facility is provided in this editor which the writers of #SUMP would have dearly loved to have available in ICL editors, namely that the editline commands do not have to be sorted by the user.

E.g. The following represents output from #XFAT annotated by a user, the job description is held elsewhere only those records from :MUSIC(4) are shown, less the comments from the compiler :

```
0007          MASTER(X)
0008          COMMON GROUND
0009          DO 1 I = 1, 100
0010          READ(5,10) Z
0011          X = X + Z
0012          Y = Y + Z * Z
0013          MEAN = X / 100.
0014          SIG = SQRT( Y /100. - MEAN*MEAN )
0015          WRITE(6,20) MEAN, SIG
0016    20    FORMAT(' MEAN = ', F10.4, '  SIGMA = ', F10.4 )
0017    C
0018    C          INSERT SUBROUTINE   GIRL WHEN IT WORKS
0019    C
0020          CALL GIRL
0021          STOP
0022          END
0023          FINISH
```

clash with X as variable

remove

missing 10 format

missing label 1

MEAN should be real

no longer needed

insert working version of subroutine from :GIRL lines 24 — 37

NB X & Y not initialised — refer to add DATA statement

The following job is now run to correct these errors, note that as the above listing is 6 greater than the line numbers in :MUSIC many of the commands use the arithmetic capabilities of #SUMP rather than the user's.

```
JOB FRED,MUSIC UPDATE,T%%%%
ED :MUSIC(4)
DE 7-6,8-6
      MASTER PIECE
      REAL MEAN
IN 21-6,1
10    FORMAT(2E0.0)
IN 12-6,1
1     CONTINUE
DE 17-6,19-6
CO 22-6,24,37,:GIRL
IN 8-6,1
      DATA X,Y/2*0./
TE
%%%%
```

It is left as an exercise for the reader to show that :MUSIC(5) holds a suitably amended version of the program.

8.0 SUMP Entities and Expressions

This chapter describes the objects used in the parameters of the basic commands, certain commands allow these parameters to be a general expression including arithmetic and logical operators. Certain reserved words refer to the state of the job and can therefore be investigated so that appropriate action may be taken.

8.1 Basic Entities

Character :- Any one of the ICL 64 character set.

Digit :- 0 to 9 inclusive

Octal Digit :- 0 to 7 inclusive

Number :- A group of digits, optionally preceded by + or -, which may be stored in a single word. I.e. an integer in the range -8388608 to 8388607. The letter K may be added to denote units of 1024.

Octal number :- A group of up to eight octal digits preceded by #.

α :- An alphabetic character, i.e. A to Z.

αnum :- An alphanumeric character in the sense that it is either α or digit (Cobol users beware).

User variable :- The character @ followed by a digit. The values of ten locations may be set and used during the course of running the job, e.g. to count the number of times we have been to the label 1YEK. They are set to zero at the start of the job. The standard macros use @8 and @9 in the current version.

Exofile :- An α followed by up to 11 further characters which may be αnum, space or -. These characters may then be followed by a generation number, which is a number in the range 0 to 4095 enclosed in round brackets. The generation number can alternatively consists of a user variable enclosed in brackets. If no generation number is specified in a command which allows this, the file with that name and the highest generation is used.

E.g. the following are all valid :-
SUBGROUPSRF7(0), ICLA-DEFAULT, A, SRL(4), ABC(@4)

Filestore name :- A filestore file is referred to in the same notation as used for an exofile, except that a colon precedes the name.

E.g. :FREDSOURCE(14), :FREDDATA

Program name :- # followed by α optionally followed by up to 3 αnum, It is recommended that four character names should be used.

E.g. #FRED, #X101

Text :- A group of up to 40 characters enclosed in single quotes. For use in an expression, e.g. to compare with the current value for ILLEGAL, up to 4 characters only may be used. If less than 4 are given, when comparing strings they are made up to 4 by addition of joker characters ←, which match any other character. Note that the character ' cannot itself be included in a string.

Label :- A digit followed by up to 11 αnum may be placed before a command or macro to show the destination of a GO command.

E.g. 1YEK DP 'PLEASE PUT FREDTAPE(3) ONLINE'

Peripheral name :- * followed by a two character mnemonic, optionally followed by a unit number in the range 0 to 15, or a user variable holding such a value. The valid mnemonics at present are

TR, TP, LP, CR, CP, DA, ED, MT, GP

E.g. *LP7, *CP, *DA@2

If the unit number is not given it is assumed to be zero. Certain commands such as LP take no account of the unit number. DA and ED are identical in all respects.

Internal value :- Any sequence of five characters starting with an exclamation mark is interpreted as a one word value, being that having a character representation given by the last 4 characters. It is used in the JD command.

System variable :- Program events can be examined by the user through the contents of certain system variables having reserved names, only the first three letters of their names need be used.

i) HALT holds up to 4 characters of a message if the core image has halted.
ii) DELETE holds up to 4 characters of a message on deletion.
iii) ILLEGAL holds a single character, as given in Appendix 2, if the core image has gone illegal.
iv) EXHAUSTED holds an internal representation of the peripheral concerned when an illegal 'T' occurs, i.e. reading off the end of a file.
v) EVENT is used to show events which cannot easily be covered by the normal program events, e.g. see FI and OF.
vi) TIME holds the time in mill seconds since the start of the job.
vii) VOLUME holds the number of records generated so far by the job.

E.g.

```
IF EXH = *CR0, GO 12A
IF ILL = 'Z' or TIM > @1, EJ
```

8.2 Operators

Expressions are built up using basic SUMP entities and the operators given below. All the operators are dyadic, i.e. are placed between two entities, except for NOT which simply precedes the logical value it negates. The alphabetically named operators can all be shortened to their first three letters.

Operation	Operator	Priority	Operand Type	Result Type
Bit pattern 'and'	MASK	11	Numeric	Numeric
Bit pattern 'or'	UNION	10	Numeric	Numeric
Bit pattern 'non-equivalence'	DIFFER	9	Numeric	Numeric
Shift left circular	SLC	8	Numeric	Numeric
Shift right logical	SRL	8	Numeric	Numeric
Multiply, Divide	*, /	7	Numeric	Numeric
Add, Subtract	+, -	6	Numeric	Numeric
Comparisons	>, <, >=, <=	5	Numeric	Logical
Equal, Unequal	=, <>	5	Numeric/Text	Logical
Logical NOT	NOT	4	Logical	Logical
Logical AND	AND	3	Logical	Logical
Logical OR	OR	2	Logical	Logical
Equivalence	IS	1	Logical	Logical

8.3 Brackets

In the absence of brackets in a complex expression, the order of evaluation is controlled by the priority of the operators. All operators of a given priority are carried out before those of a lower priority, in the case of equal priority they are evaluated in the order 'left to right'.

E.g. 7+3*2 would give 13;
7/3*2 would give 4, bearing in mid that the division uses integer arithmetic.

The order may be changed by introducing round brackets, an expression enclosed in brackets being evaluated independently.

E.g. (7+3)*2 would give 20;
7/(3*2) would give 1.

Square brackets are used to access an address in the user's current core image, the sequence [x] is replaced by the contents at address x.

E.g. [45]*2 would give twice the contents of word 45 of the core image.
[[8] mask #17777777] gives the next instruction to be obeyed, as
i) word 8 holds the address of the next instruction to be obeyed in the least significant 22 bits.
ii) [8] is this pointer.
iii) [8] MASK #17777777 isolates this address
iv) [[8] mask #17777777] is the contents of this address

N.B. Word 8 may be accessed but not altered, see AL.

Appendix 1 Glossary of terms

Actual Parameter When a user makes use of a macro, the parameters supplied by the user with the call of the macro are the actual parameters.

Binary Dump A binary dump is a copy of the Binary Object program, e.g. on cards or magnetic tape.

Binary Object Program This is the actual bit pattern resulting from the compilation and consolidation of a source program. This bit pattern is loaded into the machine and executed.

Compilation Compilation is the process of generating semi-compiled from the source program.

Compiler A program to carry out compilation.

Consolidation Is the process of combining the semi-compiled from a user source program with semi-compiled from library files, and transforming it into the binary object program, The standard program used is #XPCK. The process is also referred to as link editing.

Deadly Embrace A situation arising when two or more processes (e.g. programs) share facilities (e.g. peripherals or core store), and neither can progress because another process is holding the facility required.

Default Value When a user calls a macro and does not supply an actual parameter there may well be a value supplied by the macro definition. Where the macro supplies a value in the absence of a user supplied one, this is called the default value.

Double Buffering A programming technique in which a program assembles the current line of output while the previous line is being printed from a separate area. While this is a good idea for running under Executive, the program not being held up during the printing of the line, there is no advantage in doing this under SUMP where output is sent to disc. To avoid wasting core it is recommended that FORTRAN users should avoid LP7 and Cobol users should use NO ALTERNATE AREAS.

Dump See Binary Dump and Filestore Dump.

Edit An edit is the process of altering a file by means of edit commands.

Executive A controlling program supplied by ICL, permanently in the machine, which controls the running of all programs including SUMP.

Exofile An exofile is a disc file or a magnetic tape which is not administered by the operating system. It is entirely the responsibility of the user to safeguard an exofile from improper use.

Extracode An extracode is an assembler level instruction which cannot be carried out directly by hardware. Consequently it is the responsibility of the operating system to carry out the instructions for the object program.

Filestore Filestore is an area of disc and magnetic tape containing copies of user information such as source programs and data in such a form that it can be fed to program as card or paper tape data by the operating system. Filestore is normally organised into user areas which in turn are split into user files. A user may create and delete files in his area at will; he may also change a file by means of an edit.

Filestore Dump This is a copy of the Filestore on magnetic tape. The concept of a filestore dump is such that if the filestore is overwritten with a dump it is restored to the state which existed at the time the dump was made. More important, the dump contains copies of individual user files files at that time which may be recovered.

Formal Parameter When a macro is defined it may be necessary to refer to objects which cannot be actually specified until the macro is called by a user who supplies Actual Parameters. Inside the macro definition are objects known as Formal Parameters (%A, %B etc.) for which the actual parameters are substituted.

Hardware A general term to describe the physical components of the computer as opposed to the logical structures set up in the machine.

Instruction A set of mnemonics (e.g. LDX, BRN) exists to describe the various instructions available to programs on 1900 hardware. All instructions are one word long. In core prints under SUMP the mnemonics are given as fully as possible.

Job Description A job description is a list of commands or steering lines at the start of a source program to control the compiler. It is also applied to the commands given to the operating system, but these are called job commands in the manual.

Macro A set of stored text records from which records are generated by character substitution under the control of Actual and Default Parameters when the macro is used.

Mill An ICL term referring to the arithmetic function of the central processor, e.g. the mill time of a program is an absolute measure of the use made of the processor, rather than the real time taken for the job to run (which depends on other programs in the machine).

Object Program See Binary Object Program

Offline Not immediately accessible by a program. The term 'offlining' is used to describe the process of diverting a program's input./output to disc, auxiliary processes being used to connect this to the real peripherals when convenient.

Online Capable of being immediately accessible by a program, without any delay because of operator intervention (e.g. to find and set up a magnetic tape).

Optimising Compiler A less straightforward compiler which attempts to achieve a more efficient object program, if necessary making alterations in the logic of a source program which have no effect upon the final result. E.g. taking constant assignment statements out of a loop and carrying out one assignment only before entering the loop.

Overflow Arithmetic overflow results when the result of a calculation cannot be placed in the location reserved for it, e.g. a result outside the range -8388608 to 8388607 using integer arithmetic.

Peripheral A hardware device used for input/output. Disc and tapes are fast peripherals, card/paper-tape readers/punches and the line printer are basic peripherals.

Semi-compiled An intermediate form of parts of a program in which all machine addresses are held as offsets so that the code is not dependent on its final position in core. This form of code can be merged with library subroutines in similar form by the Consolidator to assemble a binary program.

Software A general term to describe programs and systems in the machine as opposed to the tangible parts of the computer.

Source Program A source program is the form created by the programmer before it is compiled or translated into an Object Program.

Appendix 2 Illegal Events

SUMP monitors illegal run time events by a message of the form :-

```
#name  ILL(x)    nnnn: LDX 7 1256(3)    N(M)=56299
```

where :-

name is the name of the user's program
x is an event category (see below)
nnnn is the address of the instruction causing the event.

The category of the illegal event is chosen to conform as far as possible with the standard Executive categorisations. There are 14 categories, listed below with the cause of each :-

Category	Symptom	Function
A	Unsuitable value for X field or N(M) field of instruction	152, 160, 161, 166
B	N(M) of any extracode out of range	-
C	Peripheral not allocated or attempt to doubly open a *MT or *DA file	150, 151, 152, 157
D	A transfer would result in a core violation or an attempt to run in zero core	157 165
E	Invalid message format	160
F	Undefined extracode	153, 154, 155, 162, 163, 164, 167
G	Attempt to write to a device opened in Read only mode or 'Flair' establishing a PUC outside its core area	157 -
H	'Flair' attempting to run a PUC before establishing it	-
K	Mode is undefined or *MT transfer of less than 5 words	157
O	A SUMP limit exceeded	-
W	Program has exceeded its mill limit	-
X	Attempt to address outside core (hardware detected)	-
Y	Attempt to read off the end of a filestore file	157
Z	Excess output from a job	157