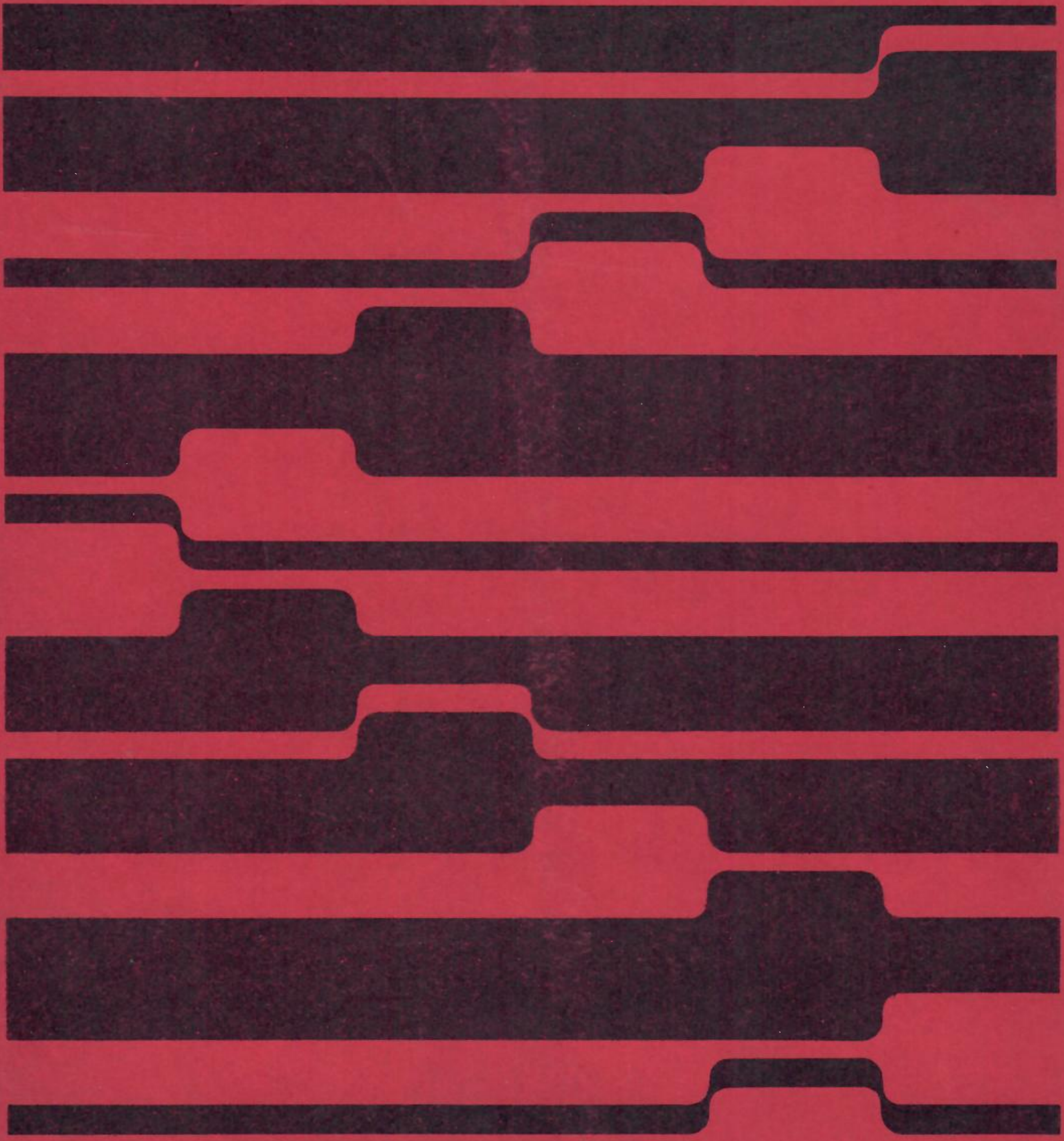
ICL

Introduction to MOP

1900 Series

4194



ICL International Computers Limited

User notice

MANUAL (NOTICE NO.) 11/3/70

4194 INTRODUCTION TO MOP (1)

File one copy of this notice with each of the manuals indicated.

FORM 1/230/45(3.69)

AMENDMENTS TO MANUAL

Page 13, line -9

This line should read:

An *expression* is formed by combining numbers, variables and functions, by means of operators. The user will

Page 26, line 9

This line should read:

```
QFORTRAN FORTPROG,,NAME
```

The second comma indicates that the Binary Program parameter is null. A parameter is said to be null if no actual parameter is specified but the required comma is inserted; if neither the parameter nor the comma is specified, the parameter is said to be omitted. In the above example, the Binary Program parameter is null; if the command had taken the form

```
QFORTRAN FORTPROG
```

the Binary Program (and Output Listing) parameters would have been omitted.

This definition of null and omitted parameters applies throughout the whole manual.

Page 37, line 16

This line should read:

If the user wants to delete his program, for example, after examining it by using the PRINT command, he can do so by giving the

Issued by Technical Publications Service, 30 - 31 Friar Street, Reading, Berkshire. RG1 1JP

Page 37, lines 14 and 15

The facility of altering word 8 should be used with care as, in certain circumstances, this word might contain more information than the address of the next instruction to be obeyed.

Page 49, line 8 of table

This line should read:

```
N(Record 2)     C'OPQ'.'STU'          S(Record 2)
```

Page 53, line 2

The categories of the monitoring file are output on the line printer and not on the log.

EDITING

Facilities exist for numbering the records to the output file and for having them listed. Each record sent to the new file can be listed. The listings are sent to the monitoring file in the LISTING category. Each record can also be numbered. The first eight character positions are used for the number, followed by the record itself. These numbers are the number of the record in the new file; they may bear little relation to the pointer since this refers to the old file, but if the new file is subsequently used as the old file for the editor the numbers become the absolute record number, that is, records are numbered from zero.

Listing and numbering are each controlled by a switch, the state of which can be altered by an editing instruction. The initial state of both switches is off, so that records are not numbered and the new file is not listed to the monitoring file.

The L instruction alters the state of the listing switch as follows:

```
L  ON    Switches it on if it is off.
         Leaves it unaltered if it is on.

L  OFF   Switches it off if it is on.
         Leaves it unaltered if it is off.

L        Switches it on if it is off.
         Switches it off if it is on.
```

The N instruction alters the state of the numbering switch in an exactly similar way.

Note: If the description of the new file given in the EDIT command includes the APPEND qualifier, the listing and numbering start from the first record to be written by the editor once the appropriate switch has been switched on.

The policy of International Computers Limited is one of continuous development and improvement of its products and services, and the right is therefore reserved to alter the information contained in this document without notice. ICL makes every endeavour to ensure the accuracy of the contents of this document but does not accept liability for any error or omission. Any equipment or software performance figures and times stated herein are those which ICL expects to be achieved in normal circumstances. Whereever practicable, ICL is willing to verify upon request the accuracy of any specific matter contained in this document.

Technical Publication 4194

First Edition January 1970

Issued by Technical Publications Service
International Computers Limited
Head Office: ICL House, Putney, London SW15
Produced by ICL Printing Services
at Letchworth, Hertfordshire

Preface

This manual provides an introduction to MOP, the multi-access system available with the ICL GEORGE 3 operating system; the description given applies to the Mark 3 version of the system.

GEORGE 3 is designed for use in installations having ICL 1900 Series computers from the 1903A upwards, provided that they have a minimum of 32K words of core store and half a million words of direct access backing store.

Chapter 1 is a general introduction to the system and describes the basic facilities available. Chapter 2 gives general hints on how to use the system, explaining how to become connected to the central computer and how to input messages. Chapter 3 describes some simple uses of the conversational language JEAN thus enabling the user to become familiar with the use of the terminal. Chapter 4 explains the method of compiling a program and Chapter 5 describes the control of a program run from the terminal. The handling of magnetic media is introduced in Chapter 6 and basic methods of editing files are explained in Chapter 7. Chapter 8 explains how to end a session at a terminal. A quick reference section is provided in Appendix 1.

Full details of the GEORGE 3 system are available in the ICL 1900 Series manual *Operating systems GEORGE 3 and 4* (Edition 3) TP 4169. Appendix 2 contains a list of all other relevant ICL 1900 Series manuals.

Contents

Chapter 1 Introduction

Chapter 2 Starting up

Chapter 3 JEAN

Chapter 7 Editing

Chapter 8 Logging out

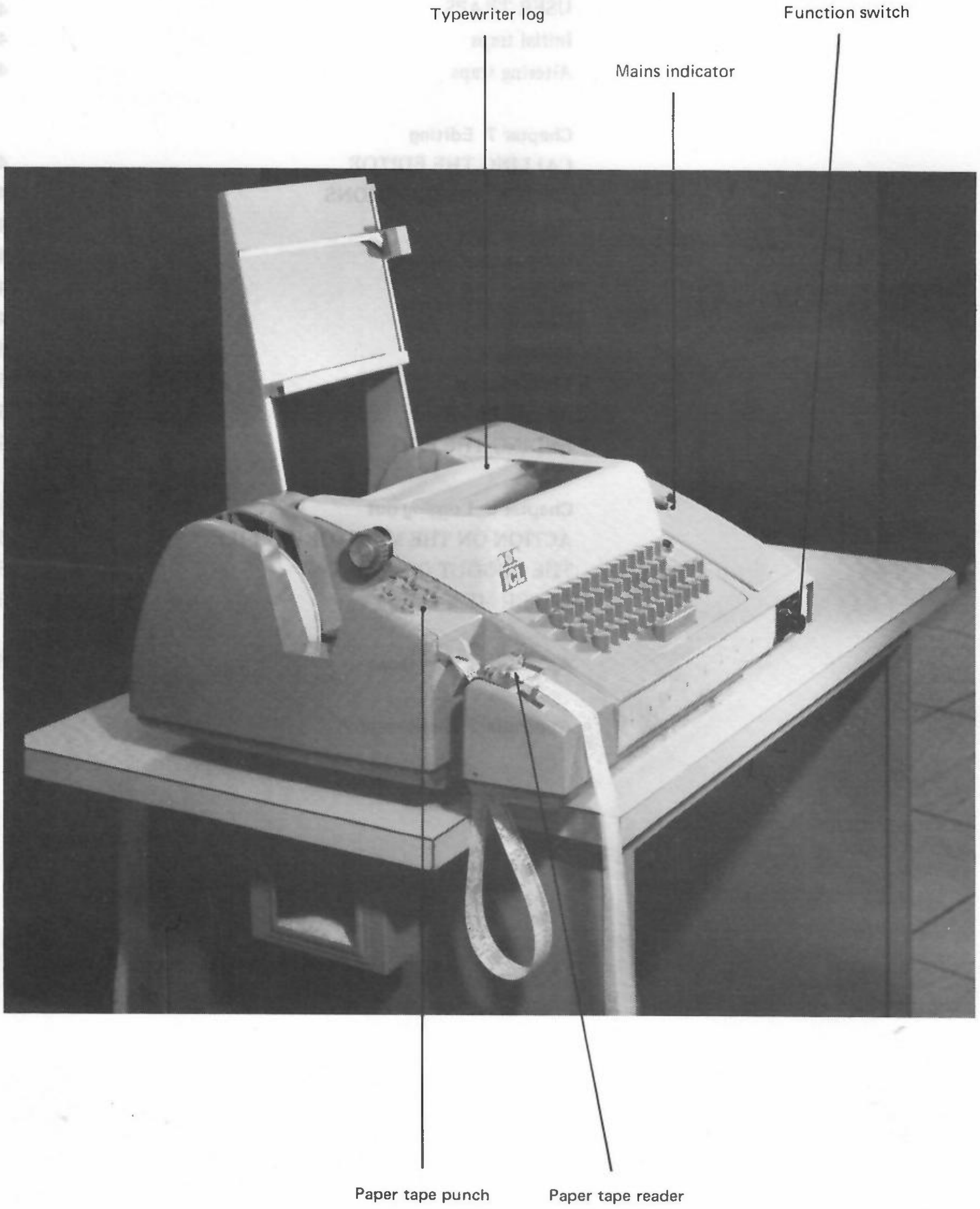


Figure 1 A 7071 Teletypewriter

Chapter 1 Introduction

MULTI-ACCESS

A multi-access system is a system in which a large number of users can 'converse' with the same computer via terminals linked directly to it. Figure 1 shows a typical terminal, in this case an ICL 7071 Teletypewriter, used for communicating with a computer. The user types messages to the computer on the keyboard and these messages are recorded on the typewriter log. The computer responds by printing out messages on the log so that the user can read them.

The multi-access system described in this manual is known as MOP, which is short for Multiple On-line Programming. MOP is part of a general system, known as GEORGE 3, used to control the operation of the larger ICL 1900 Series computers. At present the only terminals that may be used with the MOP system are 7071 and 7072 console typewriters but future versions of the system will allow the use of other terminals. A full description of the MOP system is given in the manual Operating systems GEORGE 3 and 4 which the reader should consult for further details of any facility described in this manual.

MOP

On-line

While a user is in direct communication with the computer, he is said to be *on-line* and the MOP system will keep him informed of the state of his job. For example, if his program goes illegal, a message will be printed out on the log saying that this has happened and why. The user can also communicate with the computer requesting, for example, a program to be loaded into the core store of the computer. This he does by typing messages on the keyboard.

When using his MOP terminal, a user can communicate not only with the system but also with any program that he is running from the terminal by using his typewriter to type in data or receive output from it. This makes such jobs as program testing much quicker and easier since a user is informed of the state of his program and so can see how the program handles test data.

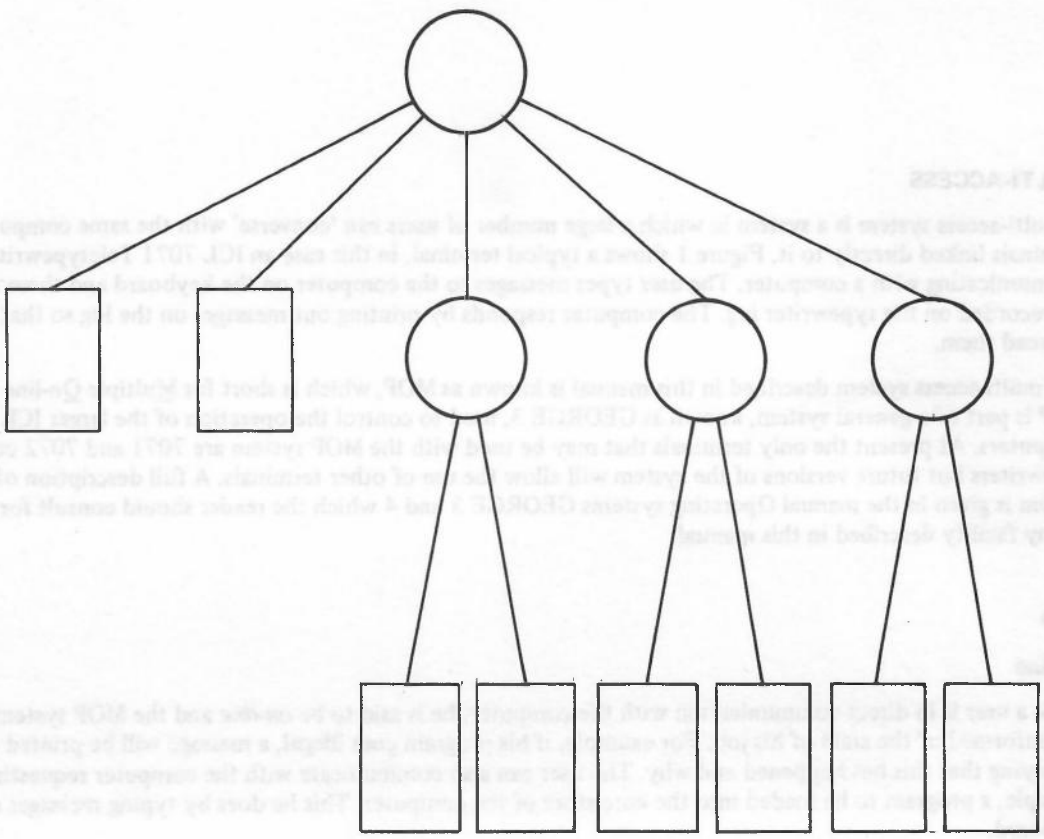
Off-line

Normally a user will want to run his job on-line, but he may disconnect it from control by the MOP terminal, that is, make it *off-line.* The job will then run under the control of GEORGE 3 subject to messages decided by the user and stored in a file. These messages say, for example, the action the system is to take should a program go illegal. Meanwhile the user can carry on with a normal MOP session at his terminal doing any work he likes until he wants to find out how the disconnected job is progressing. This he does by giving the system an appropriate signal to reconnect the job to his terminal.

Entering the system

Before anyone can use the system, he must be known to it. The system recognizes any user by his *user name* and *password,* which he has to give every time he wants to start a MOP session. Each user name is unique to the installation involved and the system keeps a file, called a *directory,* associated with each user name, in which it records details of the files belonging to that user. Directories can be created only by a user who already has a directory. When this user creates a new directory he specifies the user name that is to be associated with it and at the same time will probably also allocate budgets of, for example, processor time and magnetic tapes to the new user. Not every user may create another user but in any installation the installation manager will keep a record of who may create users and will advise any would-be user on how to acquire the requisite budgets, directory and user name.

A new directory is regarded as one of the files belonging to the user who created it. Since every user may also be the owner of *terminal* files, that is, files containing information or programs fed in by the user himself, this method of creation leads to a hierarchical structure of files as shown in Figure 2. User-created files are part of the system's



Note: Directories are indicated by circles and terminal files are indicated by squares.

Figure 2 Filestore structure

filestore, which is held on drum, disc or magnetic tape and which also contains system files created by GEORGE 3. Each file is referred to by name not location since GEORGE 3 organizes the filestore in such a way that the user has no way of knowing where a file is stored or whether GEORGE is about to move it.

Communicating with the system

When a user wants to tell the system what to do, he has to use standard messages that the system can understand and interpret. The messages that he types must be in the *GEORGE 3 command language,* the language used to communicate with GEORGE 3. This manual does not provide a full specification of this language but will give sufficient information for a user to be able to run jobs from his MOP terminal.

The GEORGE command language is made up of a series of *commands.* Each command tells the system to carry out a particular function, for example, to load a program or to allocate a certain amount of core store to it.

Editing

The MOP user may wish to alter the contents of files held in the filestore; to do this he can use the editing facilities. Chapter 7 of this manual gives an introduction to the editor and shows by examples how it may be used.

Running standard software

In the course of running his job the user may wish to use an item of standard ICL software. To do this he merely has to give an appropriate command and the system will take the requisite action. Almost any item of ICL software can be made available for use in this way. In particular, it is possible for the MOP user to take advantage of the JEAN conversational language facility, which enables him to perform arithmetical calculations on-line. An introduction to the use of JEAN is given in Chapter 3.

Chapter 2 Starting up

This chapter describes the operations that every user must carry out before he can start to run his job from his MOP terminal. The description given assumes that the terminal used is a 7071 Teletypewriter as shown in Figure 1, page viii. Messages typed on the teletype are modulated for transmission over the telephone or telegraph line. At the other end they are demodulated and passed to a terminal that converts them to a form suitable for transfer to a multiplexer; the multiplexer scans all its channels (up to 63) and transmits the messages from each channel to the central computer. When message are output by the system, they are transmitted to the multiplexer which sends them to the correct channel. The terminal then converts them to a form suitable for transfer and they are modulated for transmission. At the remote end they are demodulated and sent to the teletype where they are output.

Before attempting to start up, the user should have obtained a user name and budgets as described under *Entering the system,* page 1.

THE 7071 TELETYPEWRITER

The chief physical features of the 7071 Teletypewriter are indicated in Figure 1, page viii. The paper tape reader and punch are optional and may not be present on the actual terminal used by the reader. A full description of the teletypewriter is given in 7071 Teletypewriter Operating.

The teletypewriter may be used in two ways:

1 ON-LINE. The terminal is connected to the computer. Messages can be received from the system and transmitted to it by typing on the keyboard or running paper tape through the paper tape reader. There is also a variant of this where the messages transmitted are not printed.

2 LOCAL. The terminal is not connected to the computer. No messages can be received from the system and any messages typed on the keyboard or read on the paper tape reader will be printed locally on the typewriter log but not transmitted.

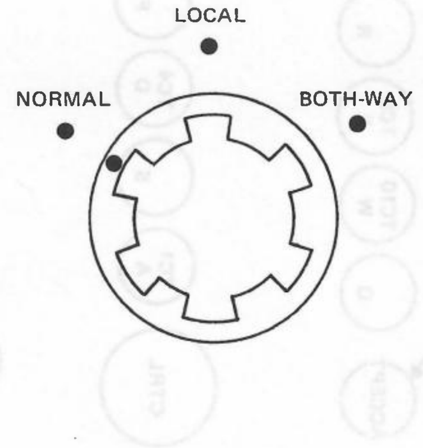
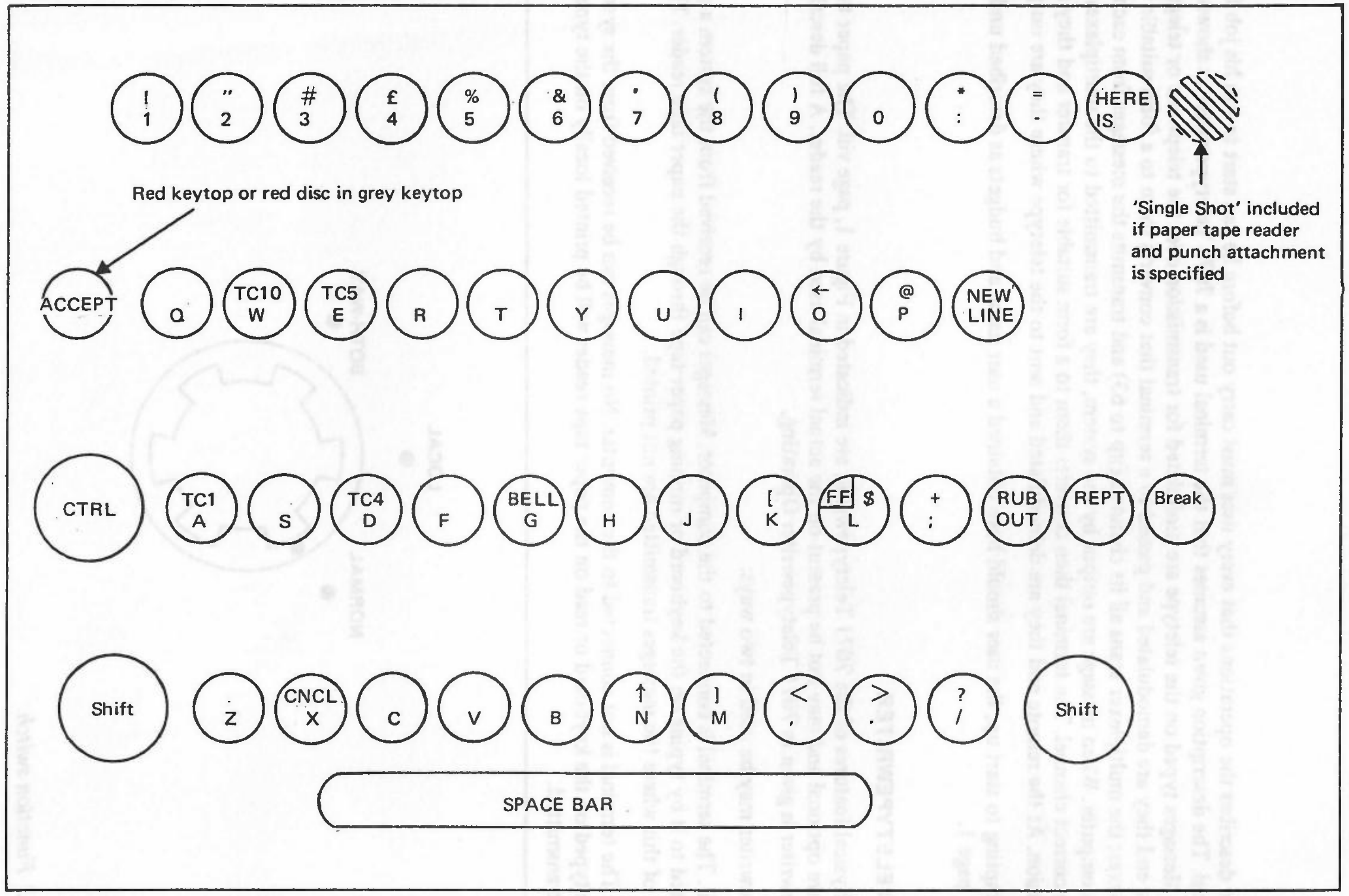


Figure 3 Function switch

Function switch

The function switch is at the right-hand end of the front nameplate cover. There are three possible positions of this switch (see Figure 3):



Red keytop or red disc in grey keytop

'Single Shot' included if paper tape reader and punch attachment is specified

Figure 4 Keyboard layout (Note: On some teletypes, the keys may differ from those shown)

NORMAL The usual on-line position. The teletypewriter will transmit and receive and print the characters transmitted and received.

LOCAL The local position. The teletypewriter will print the characters locally but will not transmit or receive.

BOTH-WAY An on-line position. The teletypewriter will transmit without printing and will print the characters received.

Keyboard

The four-row keyboard is shown in Figure 4. The keys are in the main like those on an ordinary typewriter and are used similarly. There are several keys not present on the ordinary typewriter keyboard and the use of these is described where necessary in this manual.

Mains indicators

The green mains indicator/switch is at the right-hand side of the keyboard. The lower section, which does not have a caption, is illuminated green when the mains supply is connected to the machine. The upper section, captioned START, is illuminated green while the motor is running.

Typewriter log

The typewriter log is located above the keyboard. Every message input by the user and every message output by the system may be recorded on this log. Thus it provides a useful means of reference for the MOP user.

Paper tape punch

If present, the paper tape punch is located to the left of the keyboard. It punches in standard 8-track 1900 paper tape code. The punch may be used either off-line (characters typed on the keyboard are punched into paper tape) or on-line (all characters transmitted and received are punched into paper tape).

Paper tape reader

If present, the paper tape reader is located to the left of the keyboard in front of the paper tape punch. It reads and interprets on the typewriter log standard 8-track 1900 paper tape code. The reader may be used either off-line (the characters are interpreted on the log) or on-line (the characters are transmitted and interpreted on the log).

Switching on

The user should make sure that the transmission plug is securely in its socket. The terminal should be connected to the mains supply and the mains current switched on. Both sections of the mains indicator at the right-hand side of the keyboard will then be illuminated green. During the course of a MOP session the upper half of the indicator light may go out. This indicates that the motor has stopped running; it may be restarted by pressing the upper half of the switch (labelled START). This is necessary only if the user wishes to transmit a message. If he is waiting for a message from the system, there is no need for him to restart the motor; the system itself will send a signal to do this before it sends its message to the terminal.

The user should find out from the installation manager if his terminal is directly linked to the computer or if a private switchboard system is being used. If there is a private system in operation, then he must fulfil its requirements (for example, he may have to press the BREAK key to contact the switchboard) before going on to the general switching on procedure described below.

In order to get into communication with the system, the user should:

1 Turn the function switch (see Figure 3, page 5) to the NORMAL position.

2 Press the key marked CTRL (short for CONTROL) and hold it down while pressing the A key (see Figure 4, page 6).

3 Press the key marked ACCEPT.

He should then wait for the system to output an introductory message, in the form:

```
THIS IS GEORGE 3 MARK 1.3 ON 17 JUL 69
17.06.30←
```

GENERAL HINTS

This section contains some general information about using a MOP terminal. It describes how to input a message, how to deal with mistakes, what to do if the terminal becomes disconnected from the system and what to do in the event of a breakdown.

Inputting

The symbol ← is the *invitation to type.* If the user does not type anything within approximately 60 seconds of receiving this invitation, the terminal will become disconnected from the system (see *Disconnection*, below). On receiving the invitation to type, the user sends a message to the system by typing on the keyboard. If he wishes to type a message that is longer than one line on the log, he must type a hyphen, known in this context as a *continuation character,* followed by the ACCEPT key, or a newline character; he can then continue his message on the next line. When the message is complete, he must press the ACCEPT key to send it to the system. *The ACCEPT key must be pressed after every line the user wants to send.*

Mistakes

If the user makes a mistake in typing in a message he can cancel the whole message by pressing the CTRL key and the CNCL (short for CANCEL) key (see Figure 4, page 6). The system will respond by outputting the word CANCEL on a new line followed by a left-facing arrow. This might appear on the log as:

```
        ←TUESDAY
        CANCEL←
```

The left-facing arrow is again an invitation to input a message and the user should now type the correct message.

It may be, however, that the user makes a slip only in one or two characters. In this case, he can correct the character(s) that are wrong without cancelling the entire message. If he has typed one wrong character he should type one left-facing arrow (SHIFT and letter O, see Figure 4, page 6) and this will cancel the wrong character. Similarly, two left-facing arrows will cancel two wrong characters and so on. Suppose a user types ABCDFG instead of ABCDEFG and wants to correct his mistake. He does this by typing two left-facing arrows and then EFG. This would appear on the log as:

```
        ←ABCDFG←←EFG
```

Disconnection

The MOP terminal will *time out,* that is, become disconnected from the system whenever the user does not respond to an invitation to type within approximately 60 seconds. To reconnect the terminal to the system, he should press the CTRL and A keys followed by the ACCEPT key. A suitable message, for example:

```
        RESTARTED AT 17.09.50
```

is then output and another invitation to type is issued.

Breakdowns

SYSTEM

If the system breaks down during a MOP session, the message:

```
        THE SYSTEM HAS TEMPORARILY CLOSED DOWN
```

is sent to the terminal. The user must then wait for the system to become operable again and output another introductory message before logging in to continue his job.

TERMINAL

The operator is informed whenever a MOP terminal breaks down. In addition, the job being run from the terminal is abandoned and the monitoring file, which gives details of what the job has been doing and what has happened to it (see *Monitoring,* page 33), is listed on a line printer at the installation. Should the line become operable again, the operator will inform the user who should get into communication with the system as described under *Switching on,* above.

TRANSMISSION

If a transmission error occurs while the MOP user is typing, the message:

```
        TRANSMISSION ERROR - PLEASE REPEAT
```

is output. The user is then given an invitation to type and should re-type the last line of his input.

LOGGING IN

The LOGIN command

The first step in logging in is to issue a LOGIN command (one of the commands in the GEORGE 3 command language). The user must give his user name so that the system will recognize him and also identify his job by a *job name.* For example, he might give the command:

```
LOGIN MOPJOBNO1,:ACCOUNTS
```

where MOPJOBNO1 is the job name and :ACCOUNTS is his user name (all user names start with a colon). Note that there is a space between LOGIN and the job name and that there is a comma between the job name and the user name. When starting his MOP session, the user can choose his own job name. A job name must consist of up to twelve alphanumeric, or hyphen (-) characters, with the first character alphabetic. The user should choose a job name that will mean something to him when he is referring to the log afterwards, for example MAY▽ACCTS or PROGRAMTEST.

Once the system has accepted the LOGIN command, the message:

```
TYPE PASSWORD←
```

will be output. The user must then type his *password.*

Passwords

When a user issues a LOGIN command, he gives his user name so that the system can recognize him. The system then asks him to type a password so that it can check that he really is who he claims to be. GEORGE keeps a record of all the user names and the corresponding passwords. Initially the password corresponding to a user name is set to twelve spaces so that the new user should give this as his password in response to the TYPE PASSWORD← message.

Note: There is no need to type the twelve spaces; the system will automatically add spaces to the end of what is typed so that the password contains twelve characters in all. Thus it is sufficient just to press the ACCEPT key. This will then be interpreted as a password of twelve spaces.

If the user has made a mistake in giving either his user name or password, the system will print out the message:

```
ERROR IN LOGIN : USER NAME/PASSWORD INVALID
17.08.30←
```

The second line is an invitation to type and the user should log in again, taking care to get the user name and password correct.

When the LOGIN command and the password have both been input correctly, the system will output a message to indicate that the user is now properly logged in and that his MOP session has started, for example:

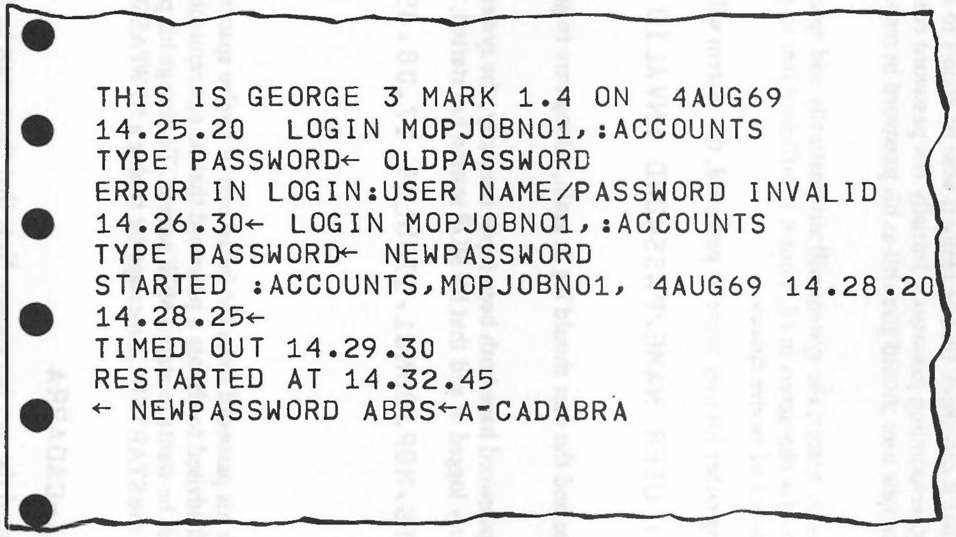
```
STARTED :ACCOUNTS,MOPJOBNO1,17JUL69 17.08.45
17.08.55←
```

GETTING A NEW PASSWORD

At this stage, the new user can change his password from the initial one of twelve spaces to any twelve characters he pleases. However, he would be well advised to choose a password that he can remember easily since he will have to give it accurately the next time he wants to start a MOP session. To change his password he should respond to the invitation to type output after the STARTED .... message by typing a NEWPASSWORD command, for example:

```
NEWPASSWORD ABRA-CADABRA
```

where ABRA-CADABRA is the password the user has chosen. Note that there must be a space between NEWPASSWORD and the password given. He should then press the ACCEPT key to transmit his command. The system will in future expect ABRA-CADABRA as the password associated with this user's name. The user may have more than one space between NEWPASSWORD and the password; the system will take the first non-space character as the first character of the password. If the user wishes, he may specify a password with less than twelve characters.



```
THIS IS GEORGE 3 MARK 1.4 ON  4AUG69
14.25.20  LOGIN MOPJOBNO1,:ACCOUNTS
TYPE PASSWORD← OLDPASSWORD
ERROR IN LOGIN:USER NAME/PASSWORD INVALID
14.26.30← LOGIN MOPJOBNO1,:ACCOUNTS
TYPE PASSWORD← NEWPASSWORD
STARTED :ACCOUNTS,MOPJOBNO1, 4AUG69 14.28.20
14.28.25←
TIMED OUT 14.29.30
RESTARTED AT 14.32.45
← NEWPASSWORD ABRS←A-CADABRA
```

Introductory message
User logs in
User types password
Message indicating mistake in user name or password
User logs in
User types password
Logging message
Invitation to type
Terminal times out because user does not reply to invitation to type
Reconnection message
User changes password (mistyping of S corrected)

Figure 5 Logging in procedure

A user can change his password at any time during a MOP session by giving a NEWPASSWORD command in response to an invitation to type. The system will change the expected password as commanded and output another invitation to type.

The section of log shown in Figure 5 illustrates the logging in procedure.

SYSTEM MESSAGES

While a MOP session is in progress, the system will output messages on the typewriter log, for example, the message:

```
        ERROR IN LOGIN :USER NAME/PASSWORD INVALID
```

mentioned earlier. As well as being printed on the log, these messages are sent to a file, called the *monitoring file,* set up by GEORGE for each job that is started. This file holds the information as a number of categories and the user can decide which categories he wants to be sent to his terminal. For example, the user will want to know if he has made a mistake in typing a command but will probably not be interested in receiving a copy of all the commands he issues. The system is preset to send to the terminal all the information except copies of the commands issued. The new user will probably not want to alter this selection. Details of the contents of the various categories of the monitoring file are given in Chapter 5 under *Monitoring,* page 33. That section also explains the use of the REPORT. command to alter the selection of information output, if this is desired.

The next chapter provides an introduction to the conversational language, JEAN, which has been designed to solve problems that can be expressed in terms of mathematics or of formal logic. The new user may find it helpful in gaining experience in using the terminal if he reads the description given and attempts to solve a few simple problems using JEAN. If the user is interested only in compiling and running programs from his terminal, he should omit Chapter 3 and go straight to Chapter 4.

Chapter 3 JEAN

This chapter provides a brief introduction to the conversational language JEAN. Enough information is given to enable the user to carry out fairly simple mathematical operations from his MOP terminal. The full JEAN language allows calculations and operations much more complex than those described in this manual. Readers requiring a full specification of JEAN should consult the JEAN manual.

THE JEAN PROGRAMS

JEAN is a *conversational language*; it has been designed with a small basic vocabulary and is easy to learn so that people can have access to the calculating and programming powers of a computer without having to learn a conventional programming language. Problems may be presented in terms of arithmetic, algebra or logic using standard mathematical or Boolean notation.

There are two methods of calculation available to the JEAN user: *programmed calculation* using indirect commands and *instant calculation* using direct commands obeyed immediately they are received by the system.

In programmed calculation, the user numbers each command as he gives it. The commands thus numbered are not implemented directly but are stored in the computer. When the user has built up a program or a subroutine in this way, he then gives a direct command to execute it. This method of calculation allows a user, for example, to test a subroutine before including it in a program; thus logic errors can be reduced and program testing time saved.

In instant calculation, each command is executed as soon as it is received by the system and the answer is printed out on the log. Only instant calculation is described in this chapter.

The JEAN system is also available in French and German versions and the user can specify the French or German program when beginning a run. These programs are identical in their operation and input language to the English version but all the system messages are output in either French or German, depending on the version being used.

Loading the program

To load the JEAN program, the user should type JEAN when he gets an invitation to type, for example:

```
17.12.30←   JEAN
```

This command will load the English version of JEAN. If he wishes to use the French or German versions, he should type JEAN F or JEAN G as appropriate. This chapter describes only the English version.

When the system has loaded the required JEAN program, the message:

```
JEAN IS READY
←
```

is output. The user can now go ahead and give a command in the JEAN language.

JEAN EXPRESSIONS

An *expression* is formed by combining numbers, variables and function, by means of operators. The user will probably want to employ arithmetic and algebraic expressions (the most commonly used types of expression) when specifying a problem. To take a very simple example, if he wants to add 1 and 2, he will specify this as 1+2. This is an example of an arithmetic expression and the plus sign, +, is an example of an arithmetic operator. When defining a JEAN expression, the user can use arithmetic operators and certain standard mathematical functions.

Functions

The user may employ the standard mathematical functions available in JEAN, for example:

SIN (X)	(where X is in radians)
COS (X)	(where X is in radians)

LOG (X)
EXP (X)
SQRT (X)

Arithmetic operators

The user can employ any of the arithmetic operators available in JEAN. These are:

+	addition
-	subtraction
*	multiplication
/	division
** or ↑	exponentiation

Order of evaluation

In JEAN expressions, functions are always evaluated first. The operators are then applied in the order:

1 Exponentiation

2 Multiplication and division

3 Addition and subraction

Operators are always applied from left to right. Parentheses may be used to make the meaning unambiguous. A part of an expression that is in parentheses is evaluated before the rest of the expression.

For example, the expression:

A*(B+C)/D↑2*SIN(E)

will be evaluated as:

$$\frac{a\,(b+c)}{d^2} \times \sin(e)$$

JEAN COMMANDS

All the messages input to JEAN take the form of JEAN commands. Each command tells the system to carry out some function and, in instant calculation, the system obeys the command and responds by typing an answer. As with the MOP system, the symbol ← is the invitation to type and the ACCEPT key must be pressed to transmit each command to the system.

The TYPE command

This command causes the system to evaluate an expression and print out the answer. For example, if, in response to an invitation to type, the user types:

```
←TYPE 2+2
```

the system will reply immediately:

```
          2+2 =          4
```

More than one expression can be evaluated by the same TYPE command. The required expressions are listed after TYPE and separated from each other by commas. For example, if the user types:

```
←TYPE 2+2,17.0+6.3,6*3,5/(2+3),SQRT(4)
```

the system will reply:

```
          2+2 =          4
     17.0+6.3 =          23.3
          6*3 =          18
      5/(2+3) =          1
      SQRT(4) =          2
```

Note that numbers can be expressed both as integers and as mixed decimal numbers. A full stop is interpreted as a decimal point.

Examples

Use JEAN to find the values of the following arithmetic expressions:

1 17×18

2 $16 + (\frac{2}{3})^2$

3 $15\frac{3}{4} - 2\frac{1}{2}$

4 $\dfrac{(1.62345 \times 3.472681)}{(6.483)^{\frac{1}{2}}}$

5 $(921)^{\frac{1}{25}}$

6 $\ln(\cos 0.42) + \ln(\sin 0.42)$.

The answers to these examples will be found at the end of this chapter, on page 18.

The SET command

This command is used to assign a value to a variable. For example, the command:

```
SET X=3.5
```

will assign the value 3.5 to the variable X and every time X is given in an expression, the system will substitute the value 3.5 before evaluating it. For example, if the user now types:

```
←TYPE X+2
```

the system will reply:

```
                    X+2 =     5.5
```

Every variable used in a TYPE command must previously have been assigned a value by a SET command. Similarly, it is permissible to include variables in an expression on the right-hand side of the equals sign in a SET command but only if they have previously been assigned values. For example, the command:

```
SET Y = LOG(X↑2)
```

is permissible only if a SET command assigning a value to X has already been given.

The following examples illustrate how the SET and TYPE commands together can be used to evaluate expressions containing variables.

```
1    ←SET X=19.47
     ←TYPE X↑2+7*X-3
        X↑2+7*X-3=   512.3709

2    ←SET Y=LOG (1.375)
     ←TYPE SQRT (Y)
        SQRT (Y) =              .5643170484
```

Examples

7 Use JEAN to evaluate the following expressions for

$x = 2, y = 17.645, z = e^{y^{\frac{1}{2}}}$

(a) $\ln(x+y)$

(b) $x^2 + 3xy + y^2$

(c) $e^y + z$

8 Do Example 7 without using the SET command.

The answers to these examples will be found at the end of this chapter, on page 18.

JEAN CLAUSES

A JEAN clause determines the conditions under which a command is obeyed. The only clause described in this manual is the FOR clause, which modifies the action of a command.

The FOR clause

The FOR clause specifies the value or values for which the command is to be obeyed and the whole command is re-obeyed for each of these values. For example, if the user types:

```
    ←TYPE X↑2+7*X-3 FOR X=11.35,2.45,6.32
```

the system will reply:

```
        X↑ 2+7*X-3  =          205.2725
        X↑ 2+7*X-3  =           20.1525
        X↑ 2+7*X-3  =           81.1824
```

Values in FOR clauses may be *stepped.* Stepped values are typed in the form:

S(*I*)*T*

where *S* is the starting value,

I is the increment and

T is the terminating value.

For example, if the user types:

```
    ← TYPE X↑2 FOR X=1(1)5
```

The system will evaluate X ↑ 2 for X = 1, 2, 3, 4 and 5 and print out the reply:

```
              X↑ 2  =          1
              X↑ 2  =          4
              X↑ 2  =          9
              X↑ 2  =         16
              X↑ 2  =         25
```

The two ways of expressing values may be mixed in one FOR clause. For example, the command:

```
    ←TYPE X↑ 2 FOR X=1,2(1)4,5
```

is equivalent to the previous TYPE command described.

Examples

9 Use JEAN to evaluate the following expressions for the values stated.

(a) ln(cos$x$) for $x = 0.2$ and 0.5

(b) $x^{1/2}$ for $x = 5,10,15$ and 20

(c) e^x for $x = 0.5,3,6,9,12$ and 20

10 Do Example 9 without using any FOR clauses.

INTERRUPTS

An interrupt occurs when the computation or output of the JEAN system is halted; JEAN types an appropriate message on the log. The user is then invited to input a command to remedy the situation.

If an error is detected in the form of a command, for example, if the user types SETX = 2 instead of SET▽X = 2, the system will respond with EH? and an invitation to type. The command should then be given in its proper form.

If an error is detected while a formula is being evaluated, an appropriate error message is given. For example, if the user gives the command:

```
←TYPE X+2
```

without having defined X, he will receive the message:

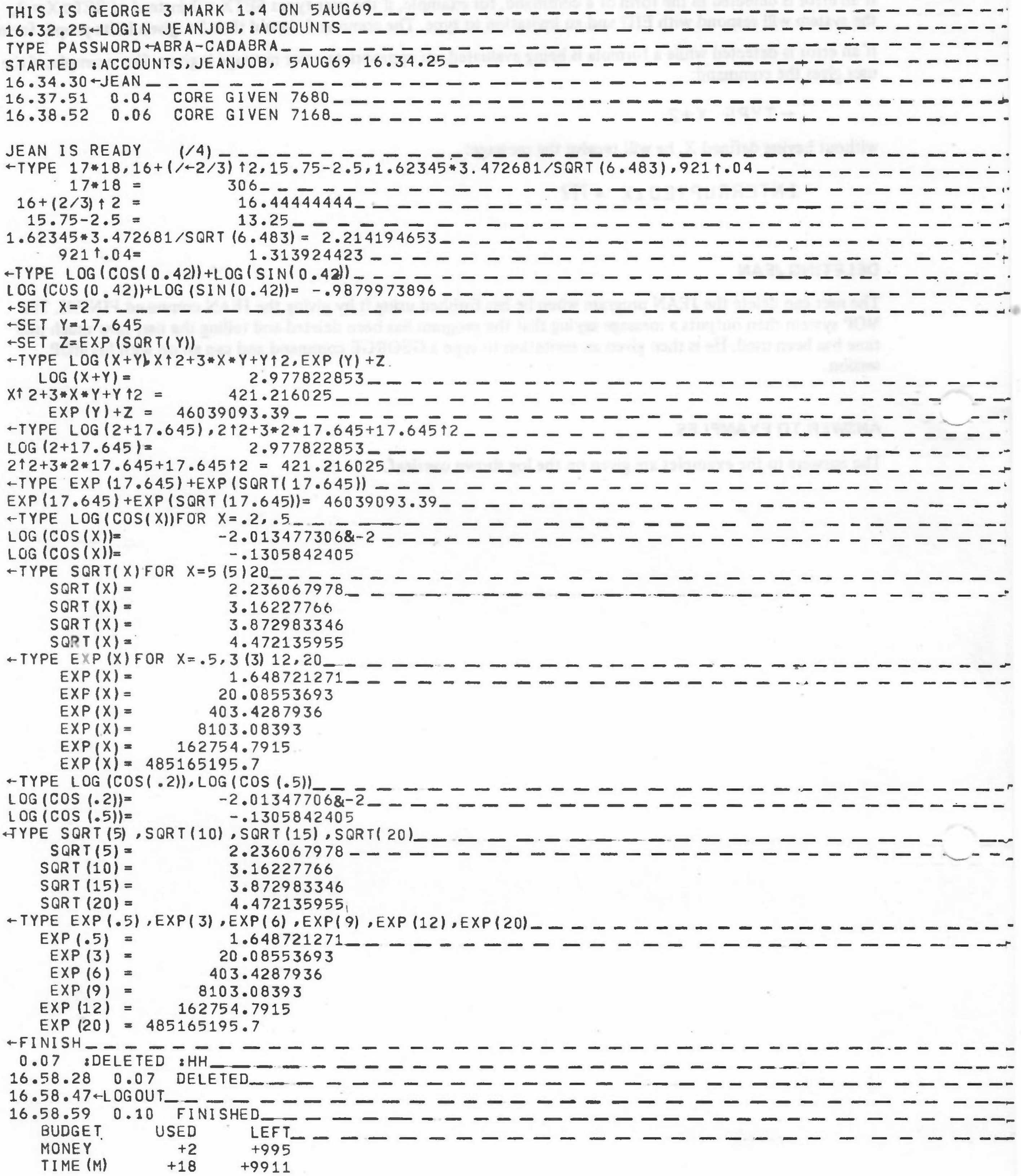
```
INTERRUPTED:X  = ???
```

DELETING JEAN

The user can delete the JEAN program when he has finished using it by giving the JEAN command FINISH. The MOP system then outputs a message saying that the program has been deleted and telling the user how much mill time has been used. He is then given an invitation to type a GEORGE command and can go on with his MOP session.

ANSWER TO EXAMPLES

The answers to the examples are given on the log shown overleaf.



```
THIS IS GEORGE 3 MARK 1.4 ON 5AUG69
16.32.25←LOGIN JEANJOB,:ACCOUNTS
TYPE PASSWORD←ABRA-CADABRA
STARTED :ACCOUNTS,JEANJOB, 5AUG69 16.34.25
16.34.30←JEAN
16.37.51  0.04  CORE GIVEN 7680
16.38.52  0.06  CORE GIVEN 7168

JEAN IS READY    (/4)
←TYPE 17*18,16+(/←2/3)↑2,15.75-2.5,1.62345*3.472681/SQRT(6.483),921↑.04
     17*18 =            306
 16+(2/3)↑2 =            16.44444444
  15.75-2.5 =            13.25
1.62345*3.472681/SQRT(6.483)= 2.214194653
     921↑.04=             1.313924423
←TYPE LOG(COS(0.42))+LOG(SIN(0.42))
LOG(COS(0.42))+LOG(SIN(0.42))= -.9879973896
←SET X=2
←SET Y=17.645
←SET Z=EXP(SQRT(Y))
←TYPE LOG(X+Y),X↑2+3*X*Y+Y↑2,EXP(Y)+Z
   LOG(X+Y)=              2.977822853
X↑2+3*X*Y+Y↑2 =         421.216025
    EXP(Y)+Z =  46039093.39
←TYPE LOG(2+17.645),2↑2+3*2*17.645+17.645↑2
LOG(2+17.645)=            2.977822853
2↑2+3*2*17.645+17.645↑2 = 421.216025
←TYPE EXP(17.645)+EXP(SQRT(17.645))
EXP(17.645)+EXP(SQRT(17.645))= 46039093.39
←TYPE LOG(COS(X))FOR X=.2,.5
LOG(COS(X))=            -2.013477306&-2
LOG(COS(X))=             -.1305842405
←TYPE SQRT(X)FOR X=5(5)20
    SQRT(X)=             2.236067978
    SQRT(X)=             3.16227766
    SQRT(X)=             3.872983346
    SQRT(X)=             4.472135955
←TYPE EXP(X)FOR X=.5,3(3)12,20
     EXP(X)=             1.648721271
     EXP(X)=            20.08553693
     EXP(X)=           403.4287936
     EXP(X)=          8103.08393
     EXP(X)=        162754.7915
     EXP(X)=     485165195.7
←TYPE LOG(COS(.2)),LOG(COS(.5))
LOG(COS(.2))=           -2.01347706&-2
LOG(COS(.5))=            -.1305842405
←TYPE SQRT(5),SQRT(10),SQRT(15),SQRT(20)
    SQRT(5)=             2.236067978
   SQRT(10)=             3.16227766
   SQRT(15)=             3.872983346
   SQRT(20)=             4.472135955
←TYPE EXP(.5),EXP(3),EXP(6),EXP(9),EXP(12),EXP(20)
   EXP(.5)  =            1.648721271
    EXP(3)  =           20.08553693
    EXP(6)  =          403.4287936
    EXP(9)  =         8103.08393
   EXP(12)  =       162754.7915
   EXP(20)  =    485165195.7
←FINISH
 0.07   :DELETED :HH
16.58.28  0.07  DELETED
16.58.47←LOGOUT
16.58.59  0.10  FINISHED
   BUDGET     USED      LEFT
   MONEY       +2       +995
   TIME(M)     +18      +9911
```

Introductory message.
User logs in.
Password given.
Logging message.
User loads JEAN.
Logging message.
Logging message.

JEAN introductory message.
Command to answer Examples 1 to 5. (Mistyping corrected.)
Answer to Example 1.
Answer to Example 2.
Answer to Example 3.
Answer to Example 4.
Answer to Example 5.
Command to answer Example 6.
Answer to Example 6.
Commands to answer Example 7.

Answer to Example 7 (a).
Answer to Example 7 (b).
Answer to Example 7 (c).
Command to answer Examples 8 (a) and 8 (b).
Answer to Example 8 (a).
Answer to Example 8 (b).
Command to answer Example 8 (c).
Answer to Example 8 (c).
Command to answer Example 9 (a).
Answers to Example 9 (a).

Command to answer Example 9 (b)
Answers to Example 9 (b).

Command to answer Example 9 (c).
Answers to Example 9 (c).

Command to answer Example 10 (a).
Answers to Example 10 (a).

Command to answer Example 10 (b).
Answers to Example 10 (b).

Command to answer Example 10 (c)
Answers to Example 10 (c).

User deletes JEAN.
Logging message.
Logging message.
User logs out.
Logging message.
Budget details.

Chapter 4 Compilation

The GEORGE 3 command language includes commands that have the effect of translating a program written in a programming language into a binary program that can be loaded into the core store of the computer. These commands are known as *compilation commands*; by means of these commands the user has access to compilers for programs written in Algol, COBOL, EMA, FORTRAN, PLAN and PLAN 4.

Each of the compilation commands allows for several alternative courses of action and these are specified in parameters by the user when issuing the command. There are parameters specifying the origin of the source program and the destination of the semi-compiled output, the binary output and the compilation listing. A full description of the facilities available with compilation commands is given in Operating systems GEORGE 3 and 4; under *Program compilation* in Chapter 8 there is a general description of the commands and full details are given in Chapter 19 both under *Program compilation macros* and under the specifications of the individual commands.

Since the compilation commands allow for several inessential options there are commands, the *Q-commands,* that provide a subset of the full facilities to speed up the compilation of programs. Since only the basic facilities are catered for in Q-commands, the implementation of these is quicker than the implementation of the full compilation commands. Only the Q-commands are described in this manual but the information given is sufficient for the user to be able to compile and run his source program.

INPUTTING SOURCE PROGRAM

If a user has a program in cards or paper tape, he can send it to the installation to be read in from a card or paper tape reader as appropriate. This method of inputting source program to a file is described in Chapter 5 under *Inputting data from a basic peripheral,* page 29. On the other hand, if the program is in a high level language such as FORTRAN, the user may wish to type it in line by line. This method of storing source program in a file is described in this section.

For example, if the user issues the command:

```
    INPUT PROGFILE
```

the system will create a terminal file called PROGFILE belonging to him. He will then receive an invitation to type consisting of a left-facing arrow and should type the first line of his source program. Once this has been transmitted, it will be stored in the file called PROGFILE and the user will receive another invitation to type. He should go on typing in the lines of his program in response to the invitations to type output by the system. When he has finished inputting the program, he should type four asterisks, ****, which the system will interpret as the terminator for the file. (Terminators are described in detail in Chapter 5 under *Inputting data from a MOP terminal* page 28.) This will appear on the log as:

```
    11.06.30← INPUT PROGFILE
    ←first line of source program
    ←second line of source program
    .
    .
    .
    ← last line of source program
    ← ****
```

The name chosen for the file must not have more than twelve characters, must start with a letter and must consist entirely of letters, digits, spaces and hyphens.

The file created will be a card file; if the user wishes input to the compiler to be read from this file, he should specify in the compilation command that the file is a card file.

It is not compulsory for the user to store his program in a file before he compiles it. One of the options in the compilation commands allows the user to type in his program line by line (as with the INPUT command) in response to invitations to type issued after the compilation command has been accepted. In this case, however, there will be no permanent record of the source program in the filestore and the user will not be able to edit it (see Chapter 7) and recompile should this be necessary; he will have to type in the whole of the corrected program. This option is described in more detail later in the chapter.

THE Q-COMMANDS

To compile an Algol, COBOL, EMA, FORTRAN, PLAN or PLAN 4 program, the user should issue a QALGOL, QCOBOL, QEMA, QFORTRAN, QPLAN or QPLAN4 command as appropriate. Each of these commands has three parameters specifying the name of the file (if any) containing the source program, the name of the file (if any) in which the binary program is to be stored and whether the listing is to be output on the log as well as on a line printer.

Each command takes the form:

verb∇parameter list

where *verb* is Q followed by the name of the language concerned, for example, QALGOL, QFORTRAN etc.

parameter list is a list of parameters, separated by commas, that define the action to be taken by the compiler.

Note that the parameters of the Q-commands must occur in a fixed sequence viz. Source, Binary, Output Listing. Parameters may be null, that is, consisting of only the comma separators. The Output Listing parameter is the only one that may be omitted (see *Examples*, page 23).

Source

FUNCTION

Specifies whether the source program is held in a file or is to be typed in from the terminal.

FORMAT

If the program is held in a file	The name of the file.
If the program is to be typed in from the terminal	The parameter is null.

EXAMPLE

SOURCEPROG	The program is held in a file called SOURCEPROG.

Binary

FUNCTION

Specifies whether the binary output is to be stored in a file or loaded into core store. If the program is overlaid, it must be stored in a file.

FORMAT

If the program is to be stored in a file	The name of a previously created disc file.
If the program is to be loaded into core store	The parameter is null.

EXAMPLE

BINPROG	The program is stored in the disc file called BINPROG, which must already exist.

Note: This parameter is always null for QPLAN 4.

Output Listing

FUNCTION

Specifies whether the user wants the listing printed on the typewriter log as well as output on a line printer.

FORMAT

If the listing is to be printed on the log	Any character or set of characters excluding - (hyphen) and % (percentage sign). The listing on the line printer is identified by the characters given.
If the listing is not to be printed on the log	The parameter is omitted.

EXAMPLE

MOPPLAN	The listing is output on the log and identified on the line printer output as MOPPLAN.

Examples

1 A MOP user has an EMA program that he wishes to compile stored in a file called EMAPROG. He wants the listing sent to his MOP terminal and the binary program loaded directly into core store. To do this he gives the command:

```
QEMA EMAPROG,,MOP
```

The system then loads the EMA compiler and outputs a logging message saying how much core store the compiler is occupying. This is followed by the compilation listing.

2 A MOP user wants to type in a PLAN4 program line by line after the compilation command has been accepted. He wants the compilation listing output on a line printer but not sent to his MOP terminal thus he issues the command:

```
QPLAN4 ,,
```

The system then loads the PLAN4 compiler and outputs a logging message saying how much core store the compiler is occupying. This is followed on a new line by an invitation to type. The user can then go ahead and type in his program. The binary program produced will be loaded directly into core store.

3 A MOP user has a PLAN4 program that he wants to compile stored in a file called PLAN4PROG. He wants the compilation listing sent to his MOP terminal. Hence he issues the command:

```
QPLAN4 PLAN4PROG,,MOPLIST
```

The system then loads the PLAN4 compiler and outputs a logging message saying how much core store the compiler is occupying. This is followed by the compilation listing.

ERROR MESSAGES

During compilation, error messages from the compilers will be printed out on the log unless the user has specified by a REPORT command (see *Monitoring*, page 33) that he does not want to receive such information at his terminal. Details of the meanings of these messages will be found in the manual relating to the language and the compiler concerned. A list of the appropriate manuals is given in the Bibliography.

COMPILERS

The compilers used in response to the compilation commands are given in the table below.

Language	*Compiler*
Algol	#XALE
COBOL	#XE20
EMA	#XMAE
FORTRAN	#XFAT
PLAN	#XPLM
PLAN4	#XPLN

Chapter 5 Running a program

When running a program from a MOP terminal, the user must tell the system what he wants done at each stage during the run. He must specify, for example, that he wants a program loaded or how he wants input and output to his program dealt with. The GEORGE 3 commands to carry out such operations are issued one by one from the terminal. When a command has been carried out by the system, an invitation to type is printed out and the user can go ahead and type the next command. This chapter describes the commands that are used to load a program, service its transfer requests, set mill (processor) time and core limits on it and enter it. In addition, there is a description of the monitoring system and its use to control the running of a program. The final section describes how to disconnect a job from control by the terminal and have it run in the background while using the terminal to run another job.

LOADING

Source programs

If a user wants to load a program that he has in source form he must input it and have it compiled before it can be loaded. He does this by issuing one of the compilation commands described in Chapter 4. If the program is loaded into core store as a result of the compilation command, it is ready to be run. At this stage, the user can take a copy of the binary program so that he can run it again without having to recompile. This is described under *Saving programs,* below. The next section deals with the program in binary form.

Binary programs

If the user has sent the binary output from compilation to a disc file or if he has stored binary program in a file by means of an INPUT command, he must give a LOAD command before he can run it, for example:

```
LOAD  BINARYPROG
```

where BINARYPROG is the name of the file in which the binary program is held. He will then receive a logging message saying how much core store the program is occupying.

SAVING PROGRAMS

Once the user has a binary program in core store, he may then want a copy of it in a file so that he can run it again without having to recompile. In this case, he should issue a SAVE command, for example:

```
SAVE  SAVEDPROG
```

where SAVEDPROG is the name of the file in which the binary program is to be held.

The user can choose his own name for the file provided it consists of no more than twelve letters, digits or hyphens and begins with a letter. Note that the file name must not include any spaces.

Note: If the user does decide that he wants to take a copy of the binary program in this way, he is advised to do it before making any peripheral connections since these connections will be lost when the program is saved and will not be restored with the program.

To load a program that has been saved in this way, the user must give a RESTORE command. (A LOAD command is not permissible because of the format of the file created by the SAVE command.) For example, the command:

```
RESTORE  SAVEDPROG
```

restores the binary program that was saved by the SAVE command above. The file SAVEDPROG will still contain a copy of the binary program.

Examples

1 A MOP user has a FORTRAN program that he wishes to compile. If there are no compilation errors, he wants a copy of the binary program as well as having it loaded into core store.

Either of the following series of commands will achieve this:

(a)
```
INPUT  FORTPROG
```
first line of program

.

.

.

last line of program
```
****
QFORTRAN  FORTPROG, NAME
SAVE  FORTSAVED
```
The FORTRAN program is input to a file called FORTPROG and compiled from there. The listing is output on the typewriter log; it will appear between the QFORTRAN and SAVE commands. The binary program produced is loaded into core store; it is then saved by the SAVE command into a file called FORTSAVED. If the user wants to load this program at any subsequent time, he must give the command:

```
RESTORE  FORTSAVED
```

(b)
```
QFORTRAN  ,DISCFORT,MOP
```
first line of program

.

.

.

last line of program
```
****
LOAD  DISCFORT
```
A disc file called DISCFORT has already been created (see Chapter 6) to hold the binary program to be produced. The compilation command is issued and the source program input line by line from the terminal. The listing is output on the typewriter log; it will appear before the LOAD command. The Binary Program parameter causes the binary program produced to be sent to the file called DISCFORT and it is loaded from there. If the user wants to load this program at any subsequent time, he must give the command:

```
LOAD  DISCFORT
```

2 A MOP user has binary program that has been input to a file called BINPROG at the installation (see *Off-lining basic peripherals,* page 27, for details). To load this program he issues the command:

```
LOAD  BINPROG
```

He then receives a logging message saying how much core store the program is occupying.

BASIC PERIPHERAL CONNECTIONS

The user can handle the input and output to his program in three ways:

1 He can input data from the MOP terminal and have the output from the program printed out on the typewriter log; this is *on-lining* the MOP terminal.

2 He can connect basic peripherals at the installation directly to the program in core so that information is input and output in a way similar to that used when running a job in a normal environment. This is not described in this manual.

3 He can input data to filestore files so that the program can access it when required and have the output from the program sent to files that he can access when he chooses. This is known as *off-lining.*

On-lining the MOP terminal

INPUT

If the user wishes to type in data to his program from a MOP terminal, he must issue an ONLINE command specifying the peripheral channel that is to expect data from the terminal. For example, the command:

```
ONLINE *CR0
```

tells the system that, whenever the program in core store issues a transfer request for card reader 0, the user is to be invited to type in data from the terminal. *CR specifies the type of peripheral as a card reader and 0 is the program unit number, used in the program to identify one particular card reader. If the user had wanted to type in data for a paper tape reader identified in his program as paper tape reader 2, he would have issued the command:

```
ONLINE *TR2
```

Note that when typing in data to service transfer requests for a paper tape reader, any shift characters that the program expects must be included as graphic characters. For example, a newline character should be typed as ↑*.

Once the program has been entered (see *Entering,* page 32), the system will output invitations to type and the user should type in one line of data in response to each of these invitations.

If the user's program has more than one basic peripheral input channel for which he wants to input data from the MOP terminal, he should include (IDENTIFY) after the *peripheral name,* that is, *CR0 or *TR2 in the examples above. The requests for input to the program will then be preceded by the peripheral name. For example, if a user gives the command:

```
ONLINE *CR0 (IDENTIFY)
```

the invitation to type data for input to the peripheral channel for card reader 0 will be given as:

```
*CR0←
```

instead of just the left facing arrow which would be the invitation to type if identification had not been specified. Clearly if there is more than one basic peripheral input channel for which data is being fed in from the MOP terminal, this identification is essential to enable the user to type in data for the correct channel.

OUTPUT

The user can direct output from the program in core to his MOP terminal. To do this, he should issue an ONLINE command specifying the name of the peripheral whose transfer requests are to be serviced by sending the output to the terminal. For example, the command:

```
ONLINE *LP1
```

tells the system that output from the line printer with program unit number as 1 is to be sent to the MOP terminal. Output from card and paper tape punch channels can also be sent to the terminal. For example, *CP0 and *TP0 identify card punch 0 and tape punch 0 respectively so the commands:

```
ONLINE *CP0

ONLINE *TP0
```

will cause output intended for card punch 0 and tape punch 0 respectively to be printed out on the typewriter log. Note that output intended for a tape punch will include shift characters.

If the user wants to direct output from more than one peripheral channel to his terminal, he should include (IDENTIFY) after the peripheral names in the ONLINE command. Each line that is printed on the typewriter log will then be preceded by the peripheral name, for example:

*LP1 *line of line printer output*

*CP0 *line of card punch output*

Off-lining basic peripherals

INPUT

If the user does not want to type in data to the program while in core, he can store it in a filestore file and then give a command that assigns this file to one of the peripheral channels of the program. Transfer requests for this channel are satisfied by reading from the file. To handle data in this way, the user must first store data in the file and then connect the file to the program. Data can be input to a file either from the MOP terminal or from a basic input peripheral at the installation; in both cases the user gives an INPUT command to store the data and an ASSIGN command to connect the file to the program.

Inputting data from a MOP terminal

The user can store data in a file from his MOP terminal in just the same way as he can store source program in a file (see *Inputting source program,* page 21). He must issue an INPUT command from his terminal to create a file and then type in the data line by line, in response to invitations to type output by the system. This might appear on the typewriter log as:

```
10.50.42← INPUT  CARDDATA
←first line of data
←second line of data
.
.
←last line of data
←****
```

Note that the system interprets four asterisks as the terminator for the file. The four asterisks are stored in the file followed by a blank record. The file thus created is suitable if the user's program expects the last line of data to be four asterisks. However, if the program expects some other four characters as the last line of data, the user can specify these characters as the terminator for the file. For example, if he chooses four hash marks as the file terminator, he must give the INPUT command as:

```
10.50.42← INPUT  CARDDATA ,S####
```

The character S specifies that the four hash marks will be stored in the file followed by a blank record.

If the user does not want the terminator to be stored, he should precede the terminator with T. Thus, if the user types:

```
10.50.42← INPUT  CARDDATA,T****
←first line of data
←second line of data
.
.
←last line of data
←****
```

The file CARDDATA will contain the data up to but not including the four asterisks.

Examples

1 INPUT CARDDATA *first line of data* . . . *last line of data* ****	A card file called CARDDATA is created; it contains all the data typed in followed by a record containing four asterisks and a blank record.
2 INPUT CARDFILE,T???? *first line of data* . . . *last line of data* ????	A card file called CARDFILE is created; it contains the data typed in but not the four question marks.

3 INPUT CARDINFO,S#### *first line of data* . . . *last line of data* ####	A card file called CARDINFO is created; it contains the data typed in followed by a record containing four hash marks and a blank record.

Inputting data from a basic peripheral

If the user wants to input data to a filestore file from a basic peripheral at the installation, he must precede the data with an INPUT command punched in a separate card or before a newline character in paper tape. In the case of an INPUT command issued in this way, the user must include his user name as the first parameter of the command with the name of the file as the second parameter.

For card files, that is, where the INPUT command is issued from a card reader, the command is otherwise identical to that which would be issued from the MOP terminal. For example, if the user's user name is :ACCOUNTS, the three commands issued at the installation that would be equivalent to the commands explained in the examples above are:

```
1 INPUT :ACCOUNTS,CARDDATA

2 INPUT :ACCOUNTS,CARDFILE,T????

3 INPUT :ACCOUNTS,CARDINFO,S####
```

If the user is inputting information to a paper tape file, he may specify the mode of the file to be created. For example, if he gives the command:

```
    INPUT :ACCOUNTS,TAPEFILE,GRAPHIC
```

the file will be in GRAPHIC mode. The modes available are:

ALLCHAR corresponding to mode #22

GRAPHIC corresponding to mode #12
NORMAL corresponding to mode #02

If no mode is specified NORMAL is assumed. Note that the data in the file must be in the mode that the user's program expects. The mode parameter must be either the third or the fourth parameter in the command. (The sequence of the terminator and mode parameters is immaterial.) For example, to create a paper tape file called TAPEDATA in mode #22 with #### as its terminator, the user types:

```
    INPUT :ACCOUNTS,TAPEDATA,ALLCHAR,S####
    first line of data
    .
    .
    .
    last line of data
    ####
```

Connecting a file to a program

Once the user has stored data in a file, he must connect the file before he enters the program. This he does by giving an ASSIGN command specifying the name of the peripheral channel and the name of the file involved. For example, the command:

```
    ASSIGN *CR0,CARDDATA
```

tells the system that transfer requests for card reader 0 are to be serviced by reading data from the file called CARDDATA.

Appropriate ASSIGN commands to connect peripherals to the files created by the INPUT commands given are described below.

Input medium	*INPUT command*	*ASSIGN command*
MOP terminal	INPUT CARDDATA	ASSIGN *CR0,CARDDATA
Card reader	INPUT :ACCOUNTS,CARDFILE	ASSIGN *CR1,CARDFILE
Paper tape reader (in mode #22)	INPUT :ACCOUNTS,TAPEFILE, ALLCHAR	ASSIGN *TR0, TAPEFILE
Paper tape reader (in mode #02)	INPUT :ACCOUNTS,TAPEDATA	ASSIGN *TR1,TAPEDATA

The file named in the ASSIGN command need not have been created by an INPUT command specifically for that ASSIGN command; it may have been created previously. In this case there are certain restrictions on the files that may be connected to the various types of peripheral channel. These are discussed under *Restrictions on files,* below.

OUTPUT

If the user wants output from his program to be sent to a filestore file instead of to his MOP terminal or an on-line peripheral, he can connect the peripheral channel to a file by means of an ASSIGN command. For example, the command:

```
ASSIGN  *LPO,OUTFILE
```

tells the system to send output intended for line printer 0 to a file called OUTFILE. When he wants to know the contents of the file later, he can give a LISTFILE command (see *Listing filestore files,* page 39) to print or punch out a copy of the file.

If the user wishes to write information at the end of an existing file, he must type (APPEND) after the file name in the ASSIGN command, for example:

```
ASSIGN *LPO,OUTFILE (APPEND)
```

RESTRICTIONS ON FILES

Input

1 The file specified in the ASSIGN command must already exist

2 If the peripheral specified in the ASSIGN command is a paper tape reader, the file must have been created by an INPUT command from a paper tape reader. The mode of the file must correspond to the mode of the instructions that are to read data from the file

3 If the peripheral specified in the ASSIGN command is a card reader, the file may have been created by one of the following:

(a) An INPUT command from a card reader

(b) An INPUT command from a paper tape reader where GRAPHIC mode has been specified

(c) An INPUT command from a MOP terminal where the data has been input in card format

(d) An ASSIGN command specifying a card punch

(e) An ASSIGN command specifying a line printer

4 The user must be allowed to read the file. Access to files is controlled by the system by means of *user traps* (see *User traps,* page 44). If the file is one the user has created himself, he will be allowed to read it unless he has given an appropriate command to alter his traps. If the file belongs to another user, the owner of the file must have given the current user permission to read it

Output

1 If the file already exists, the user must be allowed to write to it (see *User traps,* page 44); it will then be emptied and written to from the start. If the user wishes only to write additional information at the end of a file, he need not be allowed to write to it, but must be allowed to append. In either case, the owner of the file, whether the current or another user, must have given the appropriate command to allow access

2 If the file does not already exist, it will be created and written to from the start

Releasing peripherals

It is not necessary to release a peripheral channel before reallocating it. For example, the user may want to on-line his MOP terminal when his data file is exhausted. He will have given the commands:

```
INPUT  CARDDATA
```

first line of data

.
.
.

last line of data
****

```
ASSIGN  *CR0,CARDDATA
```

When the program has read all the data in CARDDATA, the next transfer request will go illegal because the file is exhausted. The message:

```
FILE  *CR0 EXHAUSTED
```

is printed out on the typewriter log followed by a logging message giving details of the amount of processor time used and the stage the program has reached, for example:

```
0.01 :FAILED :PROGRAM AT 92,,,
```

where 92 is the address of the instruction at which the program failed. The user can then give the command:

```
ONLINE *CR0
```

and resume his program, typing in the rest of the data from his terminal.

CONTROLLING THE RUN

Before entering his program, the user should consider how he wants to control its running. He can set limits on the amount of mill time and core store available to the program. The monitoring system (see *Monitoring,* page 33) will inform him of any errors in his run and he can then take what action he thinks necessary. In addition, he can use the MONITOR command to enhance the monitoring facilities. The rest of this chapter describes what conditions in the run will give rise to messages on the log and the user is advised to read this before entering his program so that he will be able to deal with whatever situations arise.

Setting a time limit

In order to avoid using up central processor time if the program goes into a loop, the user can set a limit on the amount of mill time that may be expended on the run. When this runs out, he will be informed by the message:

```
TIME UP
```

on the log. For example, the command:

```
TIME 2SECS
```

will set a limit of two seconds on the mill time of the program concerned.

SECS and MINS are the only sets of alphabetic characters allowed; if neither is given, SECS is assumed. Thus the command:

```
TIME 5
```

will set a mill time limit of five seconds.

If this command is not given, the mill time limit will be set automatically to a time determined by the installation manager.

Altering the core allocation

The user can alter the amount of core store allocated to his program at any time while the program is in core. If the user wishes to change the allocation because the core requirement is not correctly specified in the program's request slip, he must give the requisite command before the program is entered; if he wants to change the allocation while the program is running, he can give the command at any time after the program is loaded. Note that he must include an instruction in the program that outputs a message when the allocation is to be changed. This will cause a program event (see *Program events*, page 35) and give the user an opportunity to issue the command.

The CORE command alters the amount of core store available to a program. For example, if the user gives the command:

```
    CORE 4096
```

the program will be allocated 4096 words of core store. The number given in the command is always rounded up to the nearest higher multiple of 64 if it is not already a multiple of 64. If the core requested is too much for the size of the machine, the maximum core allowable is given.

Once the system has obeyed this command, a message stating the amount of core that has been given is printed out on the typewriter log.

The MONITOR command

The monitoring facilities are described under *Monitoring* and *Program events*, below. Since the MONITOR command to supplement these facilities must be issued before the program is entered, it is described here.

There are three types of MONITOR command, each fulfilling a different function. However, only two of these types are useful in a MOP environment. Details of the other can be found in Chapter 8 of Operating Systems GEORGE 3 and 4 under *Categories of program event.*

If the user wants to be informed when a program is going to delete itself, he should give the command:

```
    MONITOR ON,DELETE
```

Instead of obeying the instruction to delete itself, the program will halt and output whatever message would have been output when the program was deleted. The facility is useful if the user wants to examine the state of the program after it has run. To do this he would use the PRINT command, see *Printing out areas of core,* page 35.

If the user wants to keep a close track of the progress of a program, he can include instructions in his program that output messages to the operator at certain stages of the run. When controlling the run from a MOP terminal, the user can give himself the opportunity to issue a command whenever one of these instructions is obeyed by giving the command:

```
    MONITOR ON,DISPLAY
```

before the program is entered. The programmed message will not be output to the operator but the system will output:

```
    DISPLAY
```

on the MOP terminal and the user will be given an invitation to type a command. This facility is useful for program testing.

It is possible to switch off the effect of either of these commands by giving the command:

```
    MONITOR OFF,DELETE

 or MONITOR OFF,DISPLAY
```

as appropriate.

ENTERING

Once the user has connected all his peripheral channels and decided how he wants to control the program run, he can enter the program. To do this, he gives an ENTER command. For example, the command:

```
    ENTER 2
```

will cause the program to be entered at entry point 2. The parameter of the command indicates the entry point and the program will be entered at entry point 0 plus the number specified. If no parameter is given, the program will be entered at entry point 0.

MONITORING

Whenever a job is started with a LOGIN command, the system sets up a monitoring file to contain all the information generated by the system in the course of the job. This information comprises a number of messages which have been assigned various categories, the names and contents of which are given in the table below.

Category	*Contents*
COMERR	Command error messages, that is, information about commands that have the wrong format, are issued at the wrong time etc.
COMMANDS	Copy of each command read and lines of comment. (Comment lines are not acted on by the system; they must be preceded by #.)
COMMENT	Replies generated by some of the GEORGE 3 commands, for example, ALTER (see page 36).
DISPLAY	Messages caused by the DISPLAY, QUESTION and ANSWER commands (see page 36).
FILES	Information about changes in the filestone in the course of the job.
LISTING	Output from the LISTFILE command (see page 39) if directed to the monitoring file system and the MOP terminal.
LOGGING	Details of the amount of mill time, core and peripheral useage of the job.
OBJECT	Information generated by object programs, for example, output directed to the MOP terminal and reports of program failures.
POSTMORT	Output of the PRINT command (see page 35).

Table 1 Monitoring file categories

The user can decide for himself which categories of information he wants to receive at his terminal. The system is preset to send all the categories except COMMANDS and if a different selection of output is wanted, a REPORT command must be given. For example, the command:

```
REPORT COMERR,DISPLAY,LOGGING,OBJECT
```

will send output from the COMERR,DISPLAY,LOGGING and OBJECT categories to the terminal. Note that if output from the LISTFILE command or the PRINT command is to be received at the MOP terminal, the user must have specified that the LISTING or POSTMORT categories as appropriate of the monitoring file are to be sent to the terminal. (This will be the case if no REPORT command is given.)

The user can indicate groups of categories to be output by means of the terms ALL,ALLBUT and NONE. For example:

REPORT ALL	will cause all the categories to be output.
REPORT NONE	will suppress printing of all the categories.
REPORT ALLBUT,DISPLAY	will suppress printing of all the categories except DISPLAY.

If the user wants to know the contents of the monitoring file but does not want them printed out while he is running the job, he can specify which categories he wants listed when he logs out (see Chapter 8).

Examples of system output

Several of the messages described in Chapter 2 are sent to the monitoring file in different categories. For example:

```
STARTED :ACCOUNTS,MOPJOBNO1,17JUL69 17.08.50
```

is a LOGGING message and

```
ERROR IN LOGIN:USER NAME/PASSWORD INVALID
```

is a COMERR message.

Cause of the event	*Program state*	*Messages output*
1 Program obeys an instruction to delete itself (and also possibly to send a message to the operator).	Deleted.	Amount of mill time used followed by DELETED (and the member number if subprogramming is used) and the message (if any) that would have been output to the operator.
2 Program obeys an instruction to suspend itself and output a message to the operator.	Ready to obey the instruction after the one causing the event.	Amount of mill time used followed by HALTED (and the member number if subprogramming is used) and the message that would have been output to the operator.
3 Program obeys an instruction to delete itself (and also possibly to output a message to the operator) when a MONITOR ON, DELETE command has been given.	Ready to obey the instruction after the one causing the event.	Amount of mill time used followed by HALTED (and the member number if subprogramming is used) and the message (if any) that would have been output to the operator.
4 Magnetic tape on-line to the program has failed (see *Connecting magnetic tapes to the program,* page 42).	Ready to obey the failing instruction if the failure was detected before the transfer request was accepted; otherwise ready to obey the next instruction.	There are three messages, the last two of which are common to several program events: (a) ONLINE followed by the name of the magnetic tape peripheral that has failed followed by FAIL. (b) Amount of mill time used followed by FAILED (and the member number and values of the order number registers if subprogramming is used). (c) The failing instruction (sometimes followed by further details).
5 Allowed run time has been used.	Ready to obey the instruction after the last completed one.	There are three messages: (a) TIME UP. (b) As (b) in 4 above. (c) As (c) in 4 above.
6 Program has tried to obey a transfer request for a peripheral not allocated to it.	Ready to obey the failing instruction.	There are three messages: (a) UNALLOCATED followed by the name of the peripheral. (b) As (b) in 4 above. (c) As (c) in 4 above.
7 Program has tried to read beyond the end of a file.	Ready to obey the failing instruction.	There are three messages: (a) FILE followed by the name of the peripheral trying to read followed by EXHAUSTED. (b) As (b) in 4 above. (c) As (c) in 4 above.
8 Program has tried to write to a file that is full.	Ready to obey the failing instruction.	There are three messages: (a) OUTPUT followed by the name of the peripheral trying to write followed by FILE FULL. (b) As (b) in 4 above. (c) As (c) in 4 above.
9 Program has exceeded the limit on the number of transfer requests allowed to the file on the current run. This limit is set by the installation manager but may be altered by the user (see the specification of the ASSIGN command in Chapter 16 of Operating Systems GEORGE 3 and 4).	Ready to obey the failing instruction.	There are three messages: (a) OUTPUT followed by the name of the peripheral trying to access the file followed by LIMIT. (b) As (b) in 4 above. (c) As (c) in 4 above.
10 Any other illegal instruction in a program run.	Ready to obey the failing instruction.	There are three messages: (a) ILLEGAL followed by an identification of the cause of the event. (b) As (b) in 4 above. (c) As (c) in 4 above.
11 Program obeys an instruction to output a message to the operator when a MONITOR ON, DISPLAY command has been given.	Ready to obey the instruction after the one causing the event.	DISPLAY followed by the amount of mill time used and MONITOR (and the member number if subprogramming is used).

Table 2 Program events

PROGRAM EVENTS

A *program event* occurs if an instruction to suspend or delete the program is obeyed, if the program goes illegal or if a condition being monitored occurs. Provided the user has not suppressed reporting on the terminal of the OBJECT category of the monitoring file, he will be informed of the program event by a message on the typewriter log. He will then be invited to type a command and he should take appropriate action. For example, if the program has issued a transfer request for an unallocated peripheral channel, the user could on-line the MOP terminal and resume the program. Table 2 gives a list of the causes of program events, the resulting state of the program and an indication of the message likely to be output.

Action on program events

The user can decide for himself what action he wants to take on each program event as it arises; this is one of the reasons why working from a MOP terminal is useful. The action that is appropriate in each case depends on the cause of the event and the resulting state of the program in core. The user has several options:

1 He can try to correct the fault that caused the event by giving a GEORGE 3 command, for example, ASSIGNing a data file if the program tried to obey a transfer request for an unallocated peripheral

2 He can have specified areas of core printed out (the PRINT command) and alter the contents of specified locations (the ALTER command). This is useful if the program has encountered an illegal instruction

3 He can communicate with the operator, for example, to ask him when his MOP session ends

If the user is successful in dealing with the cause of the program event, he can then resume the program; otherwise he can delete it and carry on, for example, loading another program.

CORRECTING FAULTS

There are several situations in which the fault giving rise to the program event can be righted by giving an appropriate GEORGE command. Once this has been done, the user can resume his program as described under *Resuming the program,* page 36.

If the program has used up its allocation of mill time, the user can give it more by issuing a TIME command as described under *Setting a mill time limit,* page 31. This facility should be used with care since the program may have run out of time because it has gone into a loop. If the user suspects that this is the case, he is advised to use the PRINT command to get a print out of the program in core and check that it is correct before proceeding with the program.

If the program has tried to obey a transfer request for a peripheral that is not allocated to it, the user should allocate the relevant peripheral channel as described under *Basic peripheral connections,* page 26.

If the program has tried to read beyond the end of a file, has tried to write to a file that is full, or has exceeded the limit on the number of transfer requests allowed to the file, the user can reallocate the relevant peripheral channel by giving an ONLINE or ASSIGN command; there is no need to release the channel first. This is described under *Releasing peripherals,* page 31.

PRINTING OUT AREAS OF CORE

The user will want to have areas of core printed out if his program goes illegal. This is done by the PRINT command. For example, the command:

```
PRINT  100(20)
```

will print out 20 words of core store starting with word 100. The user can specify a number of regions in the same PRINT command. For example, the command:

```
PRINT  100(20),(200,300),400,415
```

will cause locations 100 to 119, 200 to 300, 400 and 415 to be printed out. Note the different ways of specifying which locations are to be printed. Printing is in standard format as described under the specification of the PRINT command in Chapter 16 of Operating Systems GEORGE 3 and 4.

Note: The output from the PRINT command is sent to the POSTMORT category of the monitoring file and the user will receive it on his terminal unless he has suppressed reporting of this category.

ALTERING THE CONTENTS OF A LOCATION

If the user decides to alter his program, he must give an ALTER command for each location whose contents he wants to change. For example, the command:

```
    ALTER  100,0
```

will zeroize word 100. Similarly, the commands:

```
    ALTER 100,[99]
    ALTER 99,0
```

will put the contents of word 99 into word 100 and zeroise word 99. Note that square brackets, [ ], indicate the contents of a location.

Unless he has suppressed reporting of the COMMENT category of the monitoring file, the user will receive details of the altered contents of the locations on his log, for example:

```
    X[100] = #00001240
    X[99]  = #00000000
```

COMMUNICATING WITH THE OPERATOR

The user may want to communicate with the operator, for example, to ask how much longer the MOP session will last. There are two ways in which he can do this. If he wants a reply, he should give a QUESTION command and the operator's answer will be printed out on the log. If he does not require a reply he should give a DISPLAY command. In both cases, the first parameter of the command is 1 (a routing parameter to send the text to the operator's console) and the second parameter is the text of the message (not more than 40 characters long) to be sent.

For example, the user could type:

```
    QUESTION 1, HOW MUCH LONGER OF SESSION?
```

Unless the user has suppressed reporting of the DISPLAY category of the monitoring file, he will receive the operator's answer, for example:

```
    10 MINUTES
```

In this case he cannot continue with the job until the operator replies, except by breaking in, see *Breaking in*, below.

If the user has a message that does not require an answer, he should use the DISPLAY command. For example, he might type:

```
    DISPLAY 1, PLEASE PHONE IF SESSION ABOUT TO END
```

if he wants to carry out a lengthy operation, like having a large file listed at the terminal, see *Listing filestore files*, page 39.

RESUMING THE PROGRAM

If the user has given a MONITOR ON, DISPLAY command, he will want to restart his program after each part of it has been successfully tested. In other cases, he may want to restart, for example, after altering the program. To restart, he must give a RESUME command specifying the address of the instruction at which the program is to be restarted. For example:

```
    RESUME 100
```

will cause the program to be entered at the instruction in location 100.

If the user wants to re-enter at the next instruction he should give a RESUME command with no parameter, that is:

```
    RESUME
```

DELETING THE PROGRAM

If the user cannot correct the fault, he will want to delete the program in core. To do this, he should give the command:

```
    DELETE
```

The program will then be deleted and he will receive messages to this effect on the log. He can then go ahead with another part of his job.

BREAKING IN

The user can halt his program in the middle of a run when no program event has occurred by giving a *break-in signal.* This signal consists of pressing the CTRL and A keys together followed by the ACCEPT key. He will then receive a message of the form:

```
    BREAK IN

    BROKEN IN DURING ENTER
    12.02.00←
```

This facility is useful if the user wants to examine and alter his program in core. To do this, he gives the PRINT and ALTER commands, see page 35, in response to the invitation to type.

If he wants to resume his program after breaking in in this way, the user must give a CONTINUE command. For example the command:

```
    CONTINUE
```

will resume the program from the point at which the break in occurred, that is, at the instruction whose address is held in word 8. The user can re-enter the program at another instruction by altering the contents of word 8 to the address of that instruction.

If the user wants to delete his program after examining it by using the PRINT command, he can do so by giving the QUIT command, that is:

```
    QUIT
```

The program will than be deleted and he will receive messages telling him so on the log. The QUIT command also terminates the break-in and the user receives an invitation to type another command. He can go ahead by loading another program or calling JEAN, for example.

OFF-LINE JOBS

The user may wish to run a job partly as a MOP job and partly as a background job, that is, not controlled directly from the MOP terminal. This will be the case if the MOP job involves a lengthy operation, for example, compiling a long program, not requiring any intervention by the user. Rather than waste time waiting for this operation to finish, the user can disconnect the job from the MOP terminal and allow it to run temporarily as a background job. In the meantime he can use the terminal to run another MOP job and later reconnect the first job to the terminal so that he can control it directly again.

Disconnecting a job

To disconnect the current job, the user should give a break-in signal (CTRL and A followed by ACCEPT) and then a DISCONNECT command. For example, the command:

```
    DISCONNECT  NEWMOPJOB
```

will disconnect the current job and start a new MOP job with the job name NEWMOPJOB. This new job will have a separate monitoring file and the user must give another REPORT command if he wants to alter the selection of monitoring file output from the present value of all the information except a copy of the commands. If the DISCONNECT command is given without a job name for the new job, the user will be logged out and must give a break-in signal and a LOGIN command to resume MOP operations.

For example, suppose a user issues a compilation command and then wants to disconnect this job and use JEAN while he is waiting for the compilation to finish. The log would then appear as:

```
12.02.15← INPUT PLANFILE          |  Source program stored in PLANFILE.
←first line of PLAN program       |
.                                 |
.                                 |
.                                 |
← last line of PLAN program       |
←****                             |
12.12.52← QPLAN PLANFILE,,        |
```

	Compilation command tells compiler to read source program from PLANFILE, load binary program into core and output the listing on a line printer.
User gives the break-in signal	
BREAK IN	
BROKEN IN AFTER AS IN QPLAN	Response to break-in signal tells the user that he has broken in after an ASSIGN command issued by the system as part of the QPLAN command.
12.13.15← DISCONNECT MOPJOB2	Job is disconnected, a new one started.
12.13.20← JEAN	JEAN is requested.
12.14.15 0.04 CORE GIVEN 7680	JEAN is loaded and allocated core store.
12.14.30 0.06 CORE GIVEN 7168	
JEAN IS READY	
←	The user can go ahead and use JEAN.

Reconnecting a job

When the user wants to reconnect the first job, he must give a CONNECT command specifying the job name of the first job (as given in the LOGIN command), for example:

```
CONNECT MOPJOBNO1
```

The MOP job in progress on the terminal is terminated and MOPJOBNO1 connected. Initially the connected job will be in the same state as if the user had given a break-in signal. A message in the form:

```
BROKEN IN DURING ENTER
```

will be output and the user will be given an invitation to type. He can then go ahead and control the job directly in the normal way.

A disconnected job will be suspended if it completes the command that it was obeying when it was disconnected. The job will be terminated if it is not reconnected by the user within a certain period of time determined by the installation manager. The user should make sure he connects the job again before this period runs out if he wants to continue to control it from his terminal.

Chapter 6 Further input and output

This chapter describes methods of input and output not described in Chapters 4 and 5. As described in Chapter 5 basic peripheral channels in a program are handled via the MOP terminal, on-line basic peripherals or filestore files. In the first two cases data is read (either from the terminal or from a peripheral) as the program requires it; in the last case it is stored first in filestore files and accessed as required. This chapter describes the further handling of the filestore by using the LISTFILE command to obtain a listing of a file and the ERASE command to remove all trace of a file from the filestore.

The handling of magnetic tapes and direct access devices is introduced and the connection of magnetic tape and direct access peripheral channels is described. Fuller information can be found in Chapter 10 of Operating Systems GEORGE 3 and 4.

The final section of the chapter describes the user traps by which the system keeps track of which users are allowed access to which files. Each user can grant or refuse himself or another user access to the files and tapes that belong to him.

LISTING FILESTORE FILES

The user can have a listing of any of the files that he is allowed to read by giving a LISTFILE command. If the file belongs to him and he has not given a command to withdraw READ access from himself (see *Altering traps,* page 45), he will be allowed to read it. If the file belongs to another user, that other user must have granted READ access before the current user could list the file. For example, the command:

```
LISTFILE  OUTPUTDATA
```

will cause the contents of the file called OUTPUTDATA to be sent to the LISTING category of the monitoring file and, unless reporting of this category has been suppressed, to be printed out on the log. This command could be used, for example, if the user ASSIGNed the file OUTPUTDATA to one of the output peripheral channels of his program.

The user may prefer to have the file listed on an output peripheral rather than the typewriter log especially if it is a large file because the rate of transfer to a basic peripheral is faster. In this case he must specify the type of peripheral to which the listing is to be sent, that is:

*LP for a line printer

*CP for a card punch

*TP for a tape punch

For example, the command:

```
LISTFILE  OUTPUTDATA, *LP
```

will cause the contents of the file OUTPUTDATA to be listed on a line printer.

The peripheral type specified in the LISTFILE command need not correspond to the type of the file being listed. Any type of basic peripheral file may be listed on a line printer or sent to the MOP terminal; any but line printer files may be listed on a paper tape punch; any but paper tape files may be listed on a card punch.

The user need not have the whole of a file output. If he does not want to start the listing from the beginning of a file, he should specify the number of the line at which he does want to start by including a parameter of the form:

```
FROM  30
```

which will start listing at the 30th line of the file. Similarly, if he does not want to list to the end of the file, he should include a parameter of the form:

```
LINES 100
```

which will cause listing of 100 lines of the file.

Note: If a paper tape file is to be listed on a line printer or sent to the MOP terminal, the user must include another parameter, SPECIAL, in the LISTFILE command. Thus to send the paper tape file OUTPUTDATA to the MOP terminal the user must give the command:

```
LISTFILE OUTPUTDATA,SPECIAL
```

In this case a listing of a paper tape file will include shift characters and the user should make allowance for this when trying to interpret the listing.

Examples

1 A user wants to output the whole of a paper tape file called RESULTS on a line printer. The appropriate command is:

```
LISTFILE RESULTS, *LP,SPECIAL
```

2 A user wants to output the first 50 lines of a card file called TAPEDATA on a paper tape punch. The appropriate command is:

```
LISTFILE TAPEDATA, *TP,LINES 50
```

3 A user wants the 30th to 99th lines of a line printer file called OUTPUT to be sent to his MOP terminal. The appropriate command is:

```
LISTFILE OUTPUT,FROM 30,LINES 70
```

Output format

HEADINGS

If the listing is not sent to the MOP terminal (via the monitoring file), the LISTFILE command outputs three lines of heading at the start to identify it. These are in the form:

```
#PRODUCED ON 17JUL69 AT 12.03.45
```

(This gives the date and time when the file was last written to.)

```
#OUTPUT BY LISTFILE IN : ACCOUNTS .MOPJOBNO1   ON 18JUL69 AT 17.06.30
```

(This gives the user name and the job name of the job in which the LISTFILE command was issued. The date and time indicate when the listing was started.)

```
DOCUMENT OUTPUTDATA
```

(This gives the name of the file specified in the LISTFILE command and thus identifies the listing.)

RECORDS

The exact format of the contents of the files output depends on their type and the output device on which they are listed. Details are given under the specification of the LISTFILE command in Chapter 16 of Operating Systems GEORGE 3 and 4 but it should be noted here that records more than 72 characters long will be output on more than one line of the MOP typewriter log.

ERASING FILESTORE FILES

If the user wants to erase a file that belongs to him, for example, after it has been listed, he must give an ERASE command specifying the name of the file to be erased. For example, the command:

```
ERASE MYFILE
```

will remove all trace of the file called MYFILE from the system.

The user should take care that he has no further need for the file before he gives this command because he will no longer be able to refer to it. Once a file has been erased, a new file with the same name can be created, for example, by an INPUT command, without causing any errors; it will be an entirely different file.

MAGNETIC TAPES

Magnetic tapes are used by the system as part of the backing store on which the filestore is organized. However, the user may want to ensure that a particular batch of information always stays on a particular magnetic tape so that, for example, it can be transferred easily to another installation. To enable the user to have this facility, conventional magnetic tapes may be included in the system. A record of all the tapes known to the system is kept in system files which are handled by routines known as the *librarian;* these tapes are known as *librarian tapes.* Tapes on the installation but not known to the system are known as *insecure tapes.* The user can introduce insecure tapes to the librarian and thus take advantage of the security facilities offered by the system (see *User traps,* page 44). Librarian tapes are classified into *owned tapes,* which belong to a particular user, and *pool tapes,* which are available to any user as work tapes.

This section describes how a user can acquire tapes for his own use and how to connect tapes to the program in core. Before a user can use magnetic tapes, he must have been allocated a budget of magnetic tapes, known as a SPACEMT budget, by the installation manager. If he tries to acquire a tape for his own use without having a sufficient SPACEMT budget, he will be informed of this by the system and will not be allowed to bring the tape under his control until he is given a larger allocation.

Acquiring magnetic tapes

INSECURE TAPES

To bring an insecure tape under his direct control, the user must give a NEW command specifying the serial number of the tape. For example, the command:

```
NEW (1234) (*MT) , (56213) (*MT)
```

will cause the magnetic tapes with serial numbers 1234 and 56213 to belong to the user. Note that any number of tapes may be brought under the user's control by a single NEW command and that each must be identified by its tape serial number in parenthesis followed by (*MT).

Once a NEW command has been given, the tapes specified are known by the system to belong to the user. They are known only by their serial numbers until they have been loaded at least once and the system has read the names from the tape header labels. A tape may be referred to by its tape serial number, its name or both. For example, a magnetic tape with the name MAGTAPE in its header label and with serial number 1234 can be referred to as any one of the following:

```
(1234)
MAGTAPE
(1234, MAGTAPE)
```

It should be noted that the user does not need to give a NEW command before he can connect an insecure tape to his program. Provided that the tape has a standard 1900 Series header label, an ONLINE command or transfer request in the program specifying the correct serial number of the tape will connect it to a magnetic tape peripheral channel of the program (see *Connecting magnetic tapes to the program,* below).

POOL TAPES

A pool tape is already under the control of the librarian but to acquire one for his own permanent use the user must give a GET command specifying the name to be written in the new header label. For example, the command:

```
GET  MYTAPE (*MT)
```

will cause the system to allocate one of the pool tapes to the user and write the name MYTAPE in the header label. Note that the name given must consist of not more than twelve letters, digits, hyphens or spaces beginning with a letter and that it must be followed by (*MT). If there is no pool tape available, the system asks the operator to load one.

Note: A magnetic tape name may be qualified by certain details, for example, a generation number. These details are not given here but a description of them may be found in Chapter 9 of Operating Systems GEORGE 3 and 4 under *Referring to files.*

Work tapes

If the user wishes to use a pool tape simply as a work tape, he does not have to issue a GET command for it. The appropriate ONLINE command or transfer request (see below) will allocate a tape from the pool temporarily and give it the name POOL∇TAPE.

Connecting magnetic tapes to the program

A magnetic tape can be connected to the program in core either by an ONLINE command or by an instruction in the program to open a magnetic tape. If the user issues an ONLINE command, this overrides an equivalent program instruction to open a tape. If the required tape is not loaded, the system will ask the operator to load it and will specify whether or not a write permit ring should be present; the user will also be informed by a message on the log in the form:

```
WAITING FOR MT MASTER∇TAPE
```

that his job is waiting for the tape to be loaded.

OWNED TAPES

If a user owns a tape with serial number 3604 and the name MASTER∇FILEA known to the system (that is, the tape was formerly a pool tape or has been loaded at least once since the NEW command referring to it was issued), he can connect this tape to magnetic tape peripheral channel zero of his program by any of the commands:

```
ONLINE *MT0, MASTER∇FILEA

ONLINE *MT0, (3604)

ONLINE *MT0, (3604, MASTER∇FILEA)
```

If this is the first time the tape has been loaded since the NEW command referring to it was issued, the serial number must be given in the description since the system does not yet know the name. In any case it is advisable to give the serial number because this helps the system to locate the required tape more quickly.

Note that if the user has acquired this tape by a NEW command, he must give an appropriate TRAPGO command (see *User traps,* page 44) before he can write to it since he initially has only READ access to it.

POOL TAPES

If the user wants to connect a pool tape to his program as a work tape, he can give a command in the form:

```
ONLINE *MT1
```

He will then be allocated a pool tape with a write permit ring.

If, on the other hand, the user wants to connect a pool tape to his program but also wants to become the owner of the tape, he should give a GETONLINE command. As its title suggests, this command combines the functions of the GET and ONLINE commands. For example, the command:

```
GETONLINE *MT1, MASTER∇FILEA
```

is equivalent to the two commands:

```
GET MASTER∇FILEA (*MT)

ONLINE *MT1, MASTER∇FILEA
```

Thus the effect will be to allocate a pool tape to the user, give it the name MASTER∇FILEA and connect it to the program as unit 1.

INSECURE TAPES

To connect an insecure tape to his program, the user should give an ONLINE command specifying the serial number of the tape. For example, the command:

```
ONLINE *MT1, (32765)
```

will connect the tape with serial number 32765 to the program as unit 1, provided the tape is loaded.

Releasing magnetic tape decks

Magnetic tape decks allocated to a program can be released by an instruction in the program to close the tape on the deck, by the deletion of the program to which the tape on the deck is connected or by a RELEASE command. For example, the command:

```
    RELEASE *MT2
```

will release the tape that the program is using as unit 2.

Returning magnetic tapes to the pool

If a user wishes to return an owned magnetic tape to the pool, he must give a RETURN command, for example:

```
    RETURN OLDTAPE (*MT)
```

The tape can be referred to by serial number, name or both and the description must be followed by (*MT). Since the user has a limited allocation of tapes, he will find it expedient to RETURN any tapes that he does not need.

Removing magnetic tapes from control by the system

If the user wants to remove a magnetic tape from control by the system, he must ask the operator to unload it. He should then give a RETURN command (since only pool tapes can be withdrawn from the system) and ask the operator to give the command (the DEAD command) necessary to withdraw it. Thus the following commands would be appropriate if the user wanted to remove the tape with serial number 7702 from the system's control:

```
    DISPLAY 1, PLEASE UNLOAD MT 7702
    RETURN (7702) (*MT)
    DISPLAY 1, PLEASE GIVE DEAD COMMAND FOR MT 7702
```

Loading a program from magnetic tape

The user can load a program held on a magnetic tape by giving a FIND command specifying the name of the program and identifying the tape. For example, to load the program #ABCD held on magnetic tape with serial number 76305 the user should issue the command:

```
    FIND #ABCD, MT, (76305)
```

Note that the second parameter is always MT.

If the program cannot be found on the magnetic tape specified, the user will receive an appropriate message on the log. For example, if #ABCD is not on the tape with serial number 76305, the message:

```
    #ABCD NOT FOUND ON (76305)
```

will be sent. If he wants to be sure of receiving this message, the user must not suppress reporting of the DISPLAY category of the monitoring file.

DIRECT ACCESS

Direct access files

Files in the filestore created by INPUT or ASSIGN commands are serial files. However, the user may want to create a direct access file, for example, to store the binary program produced by a compiler (page 22). A direct access file may have disc or drum format but regardless of the format, the file may be held on any part of the direct access backing store. Thus a disc file may be physically held on drum but will be organized as though it were held on disc. To create a direct access file, the user must give a CREATE command specifying what type of file he wants (E.D.S., F.D.S. or drum) and its maximum size. For example, the command:

```
    CREATE NEWFILE (*ED,KWORDS 4)
```

will set up a direct access file called NEWFILE in E.D.S. format with a size of 4K words.

The user indicates the type of file he wants by means of the characters *ED for an E.D.S. file, *FD for an F.D.S. file or *DR for a drum file. The size of the file must be given in units of 1024 words and is specified as KWORDS followed by the number of units of 1024 words. Thus KWORDS 4 indicates that the maximum size of the file is 4096 or 4K words. Note that these details of the file are enclosed in parentheses and follow the file name. They are known as *qualifiers*.

If the file that is being created is a disc file, then the user can specify how the file is to be organized by giving certain other qualifiers in the parentheses that follow the file name. For example, he can specify the bucket size, the length of the bucket header, whether the records are to be of fixed or variable length and so on. An explanation of these qualifiers is given in Chapter 14 of Operating systems GEORGE 3 and 4 under *Optional entrant description qualifiers* and the full specification of the CREATE command can be found in Chapter 16 of the same manual.

Once a direct access file has been set up in this way, the user can refer to it by name just as he would do with any other file. He can assign it to his program by an ASSIGN command or erase it with an ERASE command. Note, however, that a direct access file cannot be listed by a LISTFILE command; to get a listing the user must make use of an appropriate standard software routine (see Library Specifications).

Direct access devices

The user may wish to ensure that a particular set of information always stays on a particular direct access device. To do this he must make use of *exofiles*, that is, files on direct access devices not used as part of the GEORGE filestore; they have the same status as insecure magnetic tapes. An explanation of the use of exofiles is given in Chapter 10 of Operating systems GEORGE 3 and 4 under *Exofiles.*

Direct access peripheral channels

The user can satisfy transfer requests for the direct access peripheral channels of his program by giving ASSIGN commands connecting these channels to direct access files. A disc channel must be connected to a disc file and a drum channel to a drum file. Regardless of whether the channel being connected is for input or output, the file ASSIGNed to it must already exist. For example, if the user wants to send output from the disc channel identified as 0 in his program to a direct access file, he should issue the commands:

```
CREATE NEWLINE (*ED,KWORDS 8)

ASSIGN *ED0,NEWFILE
```

Note that the ASSIGN command has the same format as for a basic peripheral channel (see page 29). An E.D.S.. channel is identified as *ED followed by the number used to identify it in the program, an F.D.S. channel as *FD and the appropriate number and a drum channel as *DR and the appropriate number.

It is possible to satisfy transfer requests for direct access channels by on-lining an exofile. This is described in Chapter 10 of Operating systems GEORGE 3 and 4.

USER TRAPS

Every file and owned magnetic tape has one or more *user traps* associated with it. These traps are the means by which the system keeps a check on which users are allowed to access the file or tape and in what way. The modes of access are:

READ The user can read the contents of the file or tape

WRITE The user can write to the file or tape from the beginning thus overwriting the existing contents

APPEND The user can write further information to the end of the file or tape

EXECUTE The user can execute the contents of the file or tape, for example, load and enter a binary program held in a file

The user to whom the file or tape belongs controls the access of himself and all other users to it. Unless he specifically grants access to another user, he will be the only one allowed to access it in any way. Initially the system allows the user owning a file or tape access in certain modes only (see *Initial traps,* below). If he wishes to change these initial traps or to grant another user access, he must give an appropriate command (see *Altering traps,* below).

Initial traps

SERIAL FILES

When a user first opens a serial file, for example, by an INPUT or an ASSIGN command, he is allowed to write to it. However, once the file is closed after the initial writing operation, the user can only read it or execute its contents (if it is a program). These initial traps prevent the user from accidentally overwriting a file whose contents he wished to preserve.

DIRECT ACCESS FILES

When a direct access file is created (by a CREATE command), the user is allowed READ, WRITE and EXECUTE access to it. Only if he wishes to append to the file, must he change the initial traps.

OWNED MAGNETIC TAPES

If the user acquires an insecure tape for his private use by a NEW command, he is allowed READ access to it. If he acquires a pool tape by a GET or GETONLINE command, he is allowed READ and WRITE access to it.

Altering traps

THE USER'S OWN TRAPS

To give himself access to a file or tape in a mode not currently permitted, the user must give a TRAPGO command. For example, suppose a user has just introduced a tape to the librarian by the command:

```
NEW (62405) (*MT)
```

and wants to write to it. He must give the command:

```
TRAPGO (62405), WRITE
```

Note that since the user has just introduced the tape, he must refer to it by its serial number. The corresponding command for a file called DATAFILE in which data has been stored is:

```
TRAPGO DATAFILE, WRITE
```

In both these cases the contents of the tape or file could be overwritten. For example, if the user gave the command:

```
ONLINE *MT1, (62405)
```

the contents of the tape would be overwritten and if he gave the command:

```
ASSIGN *LP1, DATAFILE
```

the contents of the file would be overwritten.

If the user wants to write to a basic peripheral file without over-writing the existing contents, he should specify APPEND instead of WRITE in the TRAPGO command.

To safeguard himself from accidentally overwriting the contents of a file or tape, the user can withdraw WRITE access from himself by a TRAPSTOP command. For example, having written new information to the tape with serial number 62405 and name MASTER∇FILE, the user can preserve this information by the command:

```
TRAPSTOP MASTER∇FILE, WRITE
```

Note that the tape can now be referred to by name since it has been loaded and the system has read the header label.

OTHER USERS' TRAPS

The owner of a tape or file can grant another user access to it by giving a TRAPGO command specifying the appropriate user name. For example, the command:

```
TRAPGO DATAFILE,:PLANNING, READ
```

allows the user with user name :PLANNING to read the contents of the file called DATAFILE. Thus this user will now be allowed, for example, to have a listing (by giving a LISTFILE command) of the file if it is a serial file.

One TRAPGO command is sufficient to grant access in more than mode. For example, the command:

```
TRAPGO DATAFILE,:PLANNING,READ,APPEND
```

allows the user with user name :PLANNING access to the file called DATAFILE in READ and APPEND modes.

Similarly, to withdraw access previously granted to another user, the owner of the file or tape should give a TRAPSTOP command, for example:

```
TRAPSTOP DATAFILE,:PLANNING,APPEND
```

Chapter 7 Editing

This chapter contains a simplified description of the GEORGE 3 editor available to the MOP user. Only the basic facilities are described here and the reader is referred to Operating systems GEORGE 3 and 4 for a full specification. The user is given enough information to enable him to make straightforward insertions and deletions in an existing serial file. The file to be edited may be of any peripheral type; the editor produces a new file of the same type as the old file.

CALLING THE EDITOR

To call in the editor, the MOP user must give an EDIT command specifying the file to be edited and the name to be given to the new file that is created. For example, the command:

```
EDIT OLDFILE,RIGHTFILE
```

tells the system that the user wants to edit the file called OLDFILE and store the resulting information in a file called RIGHTFILE. The file that is to be edited must be a serial file to which the user has READ access.

When the editor has opened the files, the user receives the message:

```
EDITOR IS READY
```

on the typewriter log. He is then given an invitation to type his first instruction to the editor.

EDITING INSTRUCTIONS

The old file is edited in accordance with *editing instructions* typed by the user at the MOP terminal. Each editing instruction may specify an action to be performed on the existing file and a position within the file at which the action is to stop. For example, the user might want to transcribe part of the existing file. He then types T (to indicate transcription) and specifies where the transcription is to stop. He may then want to skip a few records so he types P (to indicate that the pointer is to be moved in the old file) and specifies the new position of the pointer.

Pointer

Position within the file is indicated by a moveable *pointer* which points at a particular character in a particular record. Before every invitation to the user to type an instruction the current position of the pointer is output in the form:

record number.character number

Records and characters are numbered from zero so that the position of the first character of the first record is 0.0. Initially the pointer is at this first character of the file and so the initial invitation to type is:

```
0.0←
```

The user may then perform some editing in the course of which the pointer is moved to the twelfth character of the sixth record, that is, character 11 of record 5. The next invitation to type is then:

```
5.11←
```

NEW POSITION OF THE POINTER

When the user wants to specify a new position of the pointer within the old file, he must give it in the form:

record.character

Using this format, there are several ways in which the user can express the parameters describing the record and the character within the record that he wants. Note that these are all different from the way in which the position of the pointer is output by the editor.

Record

The record may be specified in any of the following ways:

1 The absolute number of the record within the file; for example, #6 denotes record 6, that is, the seventh record and #16 denotes record 16, that is, the seventeenth record

2 The number of the record relative to the record at which the pointer is currently positioned (as indicated by the editor when outputting an invitation to type); for example 6 denotes the sixth record after the current one and 13 denotes the thirteenth record after the current one

3 The next record beginning with or containing a certain set of characters; for example, 'ABC' denotes the next record beginning ABC and C'ABC' denotes the next record containing the characters ABC

4 The record after the last record in the file. This is specified by giving E. This method of positioning the pointer is useful when reading or deleting as far as the end of the file (see below)

If he is specifying a record by a set of characters, the user must precede and follow the set with the same character ('in the examples above). This character is known as a *string delimiter* and must not be included in the set of characters used to identify the record; it may be any character from the set:

:;<=>?!''£%&'+/[]

Thus £ABCD£ and [ABCD[are acceptable methods of denoting the next record that starts ABCD but XABCDX and [ABCD] are not.

If the record part of the format is omitted, the editor assumes the record at which the pointer is currently positioned.

Character

The character may be specified in any of the following ways:

1 The position of the character within the record; for example, 2 denotes character 2, that is, the third character and 24 denotes character 24, that is, the twenty-fifth character in the record

2 The first of a certain set of characters; for example 'ABC' denotes the A of the next occurrence of the characters ABC in the record specified and ''FIELD∇'' denotes the F of the next occurrence of FIELD∇1'' in the record specified. Note that the characters specified are preceded and followed by a string delimiter (see *Records*, above)

3 The imaginary end of record or newline character (after the last character in the record). This is specified by E. This method of positioning the pointer is useful when reading or deleting as far as the end of a record (see below)

If the character part of the format is omitted, the editor assumes character 0, that is, the first character of the record.

Examples

This section contains some examples of how to specify a position within a file. Suppose a file contains records as shown in Figure 6.

ABCDEFG	Record 0
HIJKLM	Record 1
NOPQRSTUV	Record 2
WXYZAB	Record 3
CDEFGH	Record 4
IJKLM	Record 5
NOPQR	Record 6
STUVWXYZ	Record 7

Figure 6

The table below gives some ways of indicating a record and a character in that record. The first column indicates the character at which the pointer is pointing before the position specified in the second column is given; the third column gives the character at which the pointer will be pointing once it has been moved to the new position.

Character at which the pointer is currently positioned		*Position given*	*Character pointed at by the new position of the pointer*	
A	(Record 0)	.4	E	(Record 0)
A	(Record 0)	1.4	L	(Record 1)
F	(Record 0)	1	H	(Record 1)
H	(Record 1)	#1.'KLM'	K	(Record 1)
H	(Record 1)	0.4	L	(Record 1)
N	(Record 2)	£IJK£.3	L	(Record 5)
N	(Record 2)	#5.!KLM!	K	(Record 5)
N	(Record 2)	C'OPQ','STU'	S	(Record 2)
N	(Record 2)	C'ZAB'.2	Y	(Record 3)
I	(Record 5)	?NOP?.E	Character after R (Record 6)	
I	(Record 5)	&STUV&	S	(Record 7)
N	(Record 6)	1.4	W	(Record 7)
N	(Record 6)	E	First character of the record after Record 7	

Correcting mistakes

To correct a mistake in any of the instructions described below, the user can type F, which means forget the last line typed.

Transcribing

To transcribe part of the old file to the new file, the user types T (to indicate transcription) followed by a new position of the pointer leaving it at the character after the last character he wants to transcribe. He then presses the ACCEPT button to transcribe the part of the file from (and including) the character described by the current position of the pointer up to (but not including) the character indicated by the new position of the pointer.

For example, if the user has a file with the records as described in Figure 6, he can transcribe the first 26 characters by giving any of the following instructions in response to the invitation to type, 0.0←

```
T#3.4

T3.4

T'WXYZ'.4

TC'ZA'.'A'
```

He will then receive the invitation to type, 3.4←, and can give the next editing instruction.

If the user gives an instruction that would involve reading off the end of the old file, he will receive the message:

```
YOU'VE RUN OFF THE END OF THE FILE
```

When the editor has transcribed to the end of the file and he will then be invited to give another instruction.

Deleting

To delete part of the old file, the user types P (to indicate moving the pointer without transcribing) followed by a new position of the pointer leaving it at the character after the last character to be deleted.

For example, if the user wants to edit the file described in Figure 6 in order to produce a new file consisting of the first three and the last two records of the old file, he can achieve this by the instructions T#2.E,P#5.E and TE. This would appear on the log as:

0.0← T#3	Transcribe the whole of the first three records.
3.0← P#6.0	Move the pointer to the beginning of the seventh record, that is, record 6.
6.0← TE	Transcribe to the end of the file.

The new file would then consist of the records:

ABCDEFG
HIJKLM
NOPQR
STUVWXYZ

To produce from the original file a new file consisting of the records:

ABCDEFG
HIJAB
CDEFGH
STUVWXYZ

the user could give the instructions shown below:

0.0← T1.3	Transcribe the rest of the current record, that is, record 0, and the first three characters of the next.
3.4← T#5	Transcribe to the start of record 5.
5.0← P.E	Move the pointer past the end of the record (which has 5 characters).
5.4← TE	Transcribe to the end of the file.

Inserting

CHARACTERS

To insert a character or set of characters, the user must type I followed by the character (s) to be inserted (enclosed in string delimiters). For example, I'ABC' will insert the characters ABC before the current position of the pointer.

For example, suppose the user wants to insert the characters ABC between the fourth and fifth characters of record 3 of the file described in Figure 6; otherwise the file is to be unchanged. This could be achieved by the instructions:

0.0← T#3.4	Transcribe the first three records and the first four characters of the fourth record.
3.4← I'ABC'	Insert the characters ABC.
3.4← TE	Transcribe to the end of the file.

Record 3 of the new file would then be WXYZABCAB.

Suppose the user wants to add the characters XYZ to the end of the second record and to insert the characters ABC at the beginning of the third record. He could then give the instructions:

0.0← T1.E	Transcribe to the end of the next record.
1.5← I?XYZ?	Insert XYZ at the end of record 1.
1.5← T1	Start the next record.
2.0← I?ABC?	Insert ABC at the beginning of record 2.
2.0← TE	Transcribe to the end of the file.

RECORDS

To insert a complete record, the user should move the pointer to the beginning of the record that is to follow the new record and then give the appropriate I instruction. For example, to insert a record consisting of ABCD between the second and third records of the existing file, the following instructions could be used:

T#2.0	Transcribe the first two records.
I?ABCD	Insert ABCD before the present position of the pointer.
?	Start a new record.
TE	Transcribe to the end of the file.

If he wanted to add the characters XYZ to the end of the second record and to insert a new record, consisting of the characters ABCD between the second and third records, he could give the following instructions:

T#1.E	Transcribe to the end of the second record.
I?XYZ	Insert XYZ before the present position of the pointer.
ABCD	Start a new record with ABCD.
?	Start a new record.
TE	Transcribe to the end of the file.

Visible space

In order to make it easier for the MOP user to keep track of the number of spaces he has typed, he is allowed to issue a V instruction which allows a character to be nominated as a visible space. For example, V* tells the system to interpret an asterisk as a space when encountered within string delimiters. The character nominated as the visible space then loses its normal significance in strings.

This facility is also useful if the user wants to insert records since trailing spaces are deleted by the input routines. The V instruction makes trailing spaces significant. For example, the instructions:

```
T#2.0
I'ABCD▿▿
'
TE.0
```

will insert a record ABCD after the second record of the old file. However, the instructions:

```
V*
T#2.0
I'ABCD**
'
TE.0
```

will insert a record ABCD▿▿ after the second record of the old file.

The user can change the visible space character at any time by giving another V instruction. If V alone is typed, the editor assumes that there is now no visible space character.

BREAKING IN

If the user wishes to break in on the edit, he may do so by pressing CTRL and A followed by ACCEPT as usual. To terminate the break-in and return to the edit, he should type Z.

ENDING THE EDIT

The user can end the edit by typing E. The remainder of the old file is transcribed to the new and the edit terminated. If the user wishes to abandon the edit, he should type Q; the new file will be erased and the message:

```
EDIT ABANDONED
```

output on the log.

The user is then given an invitation to type another GEORGE 3 command and can continue with his job.

Chapter 8 Logging out

When a user has finished all the work he wanted to do from his MOP terminal, he should log out. At this stage, he can ask for some or all of the categories of the monitoring file, see page 33, to be output on the log. Since the user has no way of communicating with the system after he has logged out, it is important that he should check that he has completed the job he set out to do. If he wants, for example, to list a file that he has edited, he must give the appropriate LISTFILE command at this point. Once he has decided that he has finished, the user gives a LOGOUT command specifying the action to be taken on the monitoring file.

ACTION ON THE MONITORING FILE

When he logs out, the user should consider which (if any) of the monitoring file categories he wants output on the log. For example, the user will probably want to know how much of his budgets he has used and how much he has left but will not be interested in a list of the commands he has issued. In this case he must specify that the LOGGING category is to be output but that the COMMANDS category is to be suppressed. The required categories are specified by means of parameters of the LOGOUT command (see below). These parameters have exactly the same format as the parameters of the REPORT command (see page 33).

In addition to having its contents printed on the log, the user can have a copy of the monitoring file sent to a filestore file. This he does by giving a suitable parameter to the LOGOUT command. For example, the parameter:

```
RETAIN (COPYFILE)
```

will send a copy of the monitoring file for the present job to a serial file called COPYFILE. Note that the name of the file is enclosed in parentheses. If the user later wanted to know the contents of this file, he could give a LISTFILE command for it. The user should note that if he gets a listing of the monitoring file in this way each record will be preceded by a category code word indicating the category of the record.

THE LOGOUT COMMAND

To logout, the user gives a LOGOUT command specifying the action to be taken on the monitoring file as explained above, for example:

```
LOGOUT LISTING,POSTMORT,RETAIN(MONITORFILE)
```

If the LOGOUT command is given with no parameters, none of the categories of the monitoring file will be listed.

Examples

1 If the user gives the command:

```
LOGOUT
```

he will receive no monitoring information and no copy of the monitoring file will be taken

2 If the user gives the command:

```
LOGOUT ALL,RETAIN(COPYFILE)
```

he will receive all the information in the monitoring file and a copy of the monitoring file will be held in a file called COPYFILE

3 If the user gives the command:

```
LOGOUT LOGGING,LISTING
```

he will receive the information in the LOGGING and LISTING categories of the monitoring file and no copy of the monitoring file will be taken

OTHER MESSAGES

During a MOP session messages other than those described here may appear on the typewriter log, for example, error messages and communications from the operator. For an explanation of these the user should consult the manual Operating systems GEORGE 3 and 4.

Appendix 1 Reference

This appendix contains a description of all the parameters and all the commands introduced in the manual; it is intended as a quick reference section. It should be noted, however, that the descriptions given here do not constitute full specifications; for these the reader must consult the manual Operating Systems GEORGE 3 and 4. In addition, a brief outline is given of the function of the other GEORGE 3 commands, relevant to the MOP user but not described in this introduction.

First of all the parameter formats are described; the parameters are arranged in alphabetical order. The commands introduced in the manual are then specified by giving their name (and its shortened form, which may be used instead), a brief outline of their function, a picture of their format and a reference to their explanation in the text of the manual. Finally the other GEORGE 3 commands are listed and their function briefly outlined. Within each section the commands are arranged in alphabetical order.

PARAMETERS

Access mode

This parameter is one of the following: READ, WRITE, APPEND, EXECUTE.

Action on monitoring file

This parameter consists of a list of the desired categories of the monitoring file, individual categories being separated by commas. The categories may be grouped by using ALL (the whole file is wanted), ALLBUT (the whole file except the categories that follow ALLBUT is wanted) or NONE (none of the file is wanted).

Event type

This parameter is either DELETE or DISPLAY.

File name

This parameter consists of up to twelve letters, digits, spaces or hyphens, beginning with a letter; leading spaces are ignored and trailing spaces have no effect since the system space fills the name up to twelve characters. If the name is the name of a saved file, it must not contain any spaces.

Job name

This parameter consists of up to twelve letters, digits or hyphens, beginning with a letter.

Magnetic tape description

This parameter may be given in one of three ways:

1 (*tape serial number*)

2 *magnetic tape name*

3 (*tape serial number, magnetic tape name*)

For the formats of tape serial numbers and magnetic tape names see below under the appropriate headings.

Magnetic tape name

This parameter consist of twelve letters, digits, spaces or hyphens, beginning with a letter.

Mill time

This parameter consists of a number followed by SECS or MINS as appropriate. If neither SECS nor MINS is given, SECS is assumed.

Mode

This parameter may be one of the following:

ALLCHAR	meaning paper tape mode #22.
GRAPHIC	meaning paper tape mode #12.
NORMAL	meaning paper tape mode #02.
CARDS	meaning line image (0).

If the mode parameter is omitted, NORMAL mode is assumed from a paper tape file and CARDS mode from a card file or the MOP terminal.

Number

This parameter may be any of the following:

1 A decimal number not greater than 8388607

2 An octal number (indicated by #) less than #40000000

3 The contents of a location (indicated by [ ])

4 An expression formed by combining any of the above three formats by means of the operators + (plus), - (minus), * (multiplied by) and / (divided by)

Peripheral name

This parameter consists of the appropriate peripheral type (see below) followed by a decimal integer less than 64 (corresponding to the integer used in the program to identify the peripheral channel). If the integer is omitted, it is assumed to be zero.

Peripheral type

This parameter consists of an asterisk followed by a two-character code identifying the peripheral. A list of peripheral types is given in Table 3 below.

Peripheral type	*Description*
* TR	Paper tape reader
* TP	Paper tape punch
* LP	Line printer
* CR	Card reader
* CP	Card punch
* MT	Magnetic tape
* ED	Exchangeable disc
* DR	Magnetic drum
* FD	Fixed disc

Table 3: Peripheral types

Program name

This parameter consists of # followed by the four-character program name.

Region

This parameter can be specified in one of three ways:

1 As (*n*,*m*) where *n* and *m* are numbers specifying the first and last locations of the region respectively

2 As *n*(*m*) where *n* is a number specifying the first location of the region and *m* is a number giving the total number of words to be printed

3 As *n* where *n* is a number specifying a location to be printed

Tape serial number

A tape serial number is an octal integer less than #40000000; it is not preceded by # and must be enclosed in parentheses.

Terminator

This parameter consists of T or S followed by the terminator chosen by the user for the input document. If the terminator is less than four non-space characters long, spaces are inserted at the end and the terminator thus formed will be assumed in the input document. For cards, the terminator may be any four characters not including a comma; space characters are ignored. For paper tape, the four characters must be α shift or α/β shift characters, this excludes lower case letters, the underline character, the first national character, $,], ↑ and ←.

Text

This parameter may consist of any of the characters that can be generated using the MOP terminal.

User name

This parameter consists of a colon followed by up to twelve letters, digits, spaces or hyphens, beginning with a letter; spaces before the first letter are ignored.

Version

This parameter is E for English, F for French and G for German. If it is omitted, E is assumed.

COMMANDS DESCRIBED IN THE MANUAL

ALTER (AL)

Resets the contents of a location in the program in core.

ALTER *number,number*

See *Altering the contents of a location,* page 36.

ASSIGN (AS)

Causes a filestore file to be opened and associated with the current program in such a way that transfer requests for a specified peripheral are serviced by reading from or writing to the file.

ASSIGN *peripheral name,file name*

See *Connecting a file to a program,* page 29.

CONNECT (CN)

Connects a background job to the MOP terminal.

CONNECT *job name*

See *Reconnecting a job,* page 38.

CONTINUE (CU)

Terminates a break-in and continues a job from the point at which the break-in occurred.

CONTINUE

See *Breaking in,* page 37.

CORE (CO)

Alters the amount of core store available to an object program.

CORE *number*

See *Altering the core allocation,* page 31.

CREATE (CE)

Creates a direct access file in the filestore.

CREATE *file name*

See *Direct access files,* page 43.

DELETE (DL)

Deletes the program currently in core.

DELETE

See *Deleting the program,* page 36.

DISCONNECT (DC)

Disconnects the current MOP job from the terminal and continues it as a background job.

DISCONNECT *job name*

See *Disconnecting a job,* page 37.

DISPLAY (DP)

Sends a message to the monitoring file system and the operator's console.

DISPLAY 1,*text*

(The text may be up to 40 characters.)

See *Communicating with the operator,* page 36.

EDIT (ED)

Calls in the editor.

EDIT *file name,file name*

See *Calling the editor,* page 47.

ENTER (EN)

Enters the current program at the specified entry point.

ENTER *number*

See *Entering,* page 32.

ERASE (ER)

Removes all trace of a file from the filestore.

ERASE *file name*

See *Erasing filestore files,* page 40.

FIND

Loads the named program from the specified magnetic tape.

FIND *program name,MT,magnetic tape description*

See *Loading a program from magnetic tape,* page 43.

GET (GE)

Allocates a magnetic tape currently in the pool to the user and relabels it.

GET *magnetic tape name*(*MT)

See *Pool tapes,* page 41.

GETONLINE

Takes a magnetic tape from the pool, relabels it and makes it on-line to the current program.

GETONLINE *magnetic tape peripheral name,magnetic tape name*

See *Pool tapes,* page 42.

INPUT (IN)

Reads the lines immediately following into a serial file.

1 INPUT *user name,file name,mode,terminator* (used when inputting from a basic peripheral)

2 INPUT *file name,mode,terminator* (used when the user is known to the system)

See *Off-lining basic peripherals*, page 27.

JEAN

Runs GEORGE 3 JEAN.

JEAN *version*

See *Loading the program,* page 13.

LISTFILE (LF)

Outputs part or all of a serial file on a basic peripheral or writes it to the monitoring file system (and thence to the MOP terminal).

LISTFILE *file name,peripheral type,* FROM *number,* LINES *number,* SPECIAL

See *Listing filestore files,* page 39.

LOAD (LO)

Loads into core the binary program contained in the named file.

LOAD *file name*

See *Binary programs,* page 25.

LOGIN (LN)

Starts a MOP job.

LOGIN *job name,user name*

See *Logging in,* page 9.

LOGOUT (LT)

Terminates a MOP job.

LOGOUT *action on monitoring file,*RETAIN(*file name*)

See *The LOGOUT command,* page 53.

MONITOR (MN)

Starts or stops the monitoring of program events of a specified type.

1 MONITOR ON,*event type*

2 MONITOR OFF,*event type*

See *The MONITOR command,* page 32.

NEW (NE)

Brings one or more magnetic tapes under the user's control.

NEW *tsn,tsn, ... tsn*

See *Acquiring magnetic tapes,* page 41.

NEWPASSWORD (NP)

Changes the user's password.

NEWPASSWORD *text*

(The text may be up to twelve characters).

See *Getting a new password,* page 9.

ONLINE (OL)

Connects a magnetic tape on-line to the current program.

ONLINE *peripheral name,magnetic tape description*

See *Connecting magnetic tapes to the program,* page 42.

PRINT (PT)

Sends one or more regions of the program currently in core to the monitoring file system (and possibly to the MOP terminal dependent upon the current reporting).

PRINT *region,region ... region*

See *Printing out areas of core,* page 35.

QALGOL, QCOBOL, QEMA, QFORTRAN, QPLAN, QPLAN4

Compile programs written in Algol, COBOL, EMA, FORTRAN, PLAN and PLAN 4 respectively.

```
QALGOL    )
QCOBOL    |
QEMA      | file name,file name,text
QFORTRAN  | (The text excludes the characters % and -).
QPLAN     |
QPLAN 4   )
```

See *The Q-commands,* page 22.

QUESTION (QN)

Sends a question to the monitoring file and the operator's console.

QUESTION 1,*text*

(The text may be up to 40 characters long.)

See *Communicating with the operator,* page 36.

QUIT (QU)

Cancels a break-in and deletes the program currently in core.

QUIT

See *Breaking in,* page 37.

RELEASE (RL)

Releases a peripheral from the current program.

RELEASE *peripheral name*

See *Releasing magnetic tape decks,* page 43.

REPORT (RP)

Alters the selection of the monitoring file to be output on the MOP terminal during a MOP job.

REPORT *action on monitoring file*

See *Monitoring,* page 33.

RESTORE (RS)

Loads a program from a file in which it has been saved.

RESTORE *file name*

See *Saving programs,* page 25.

RESUME (RM)

Restarts the current program at the next instruction or at the location specified.

RESUME *number*

See *Resuming the program,* page 36.

RETURN (RT)

Returns a magnetic tape to the pool.

RETURN *magnetic tape description*(*MT)

See *Returning magnetic tapes to the pool,* page 43.

SAVE (SV)

Saves the current program in a file.

SAVE *file name*

(The file name must not contain any spaces.)

See *Saving programs,* page 25.

TIME (TI)

Sets a limit on the mill time allowed to the current program.

TIME *mill time*

See *Setting a time limit,* page 31.

TRAPGO (TG)

Permits a user access to a file or magnetic tape.

1 TRAPGO *file name,user name,access mode,access mode* ... (used for file traps)

2 TRAPGO *magnetic tape description,user name,access mode, access mode* ... (used for magnetic tape traps)

See *Altering traps,* page 45.

TRAPSTOP (TS)

Withdraws from a user access to a file or magnetic tape.

1 TRAPSTOP *file name,user name,access mode,access mode* ... (used for file traps)

2 TRAPSTOP *magnetic tape description,user name,access mode,access mode* ... (used for magnetic tape traps)

See *Altering traps,* page 45.

OTHER GEORGE COMMANDS

BUDGETQUERY	–	enables a user to find out the budgets at his disposal.
COPY	–	makes a copy of an existing terminal file.
COPYIN	–	copies subfiles from a magnetic tape into card files in the filestore.
COPYOUT	–	copies basic peripheral files from the filestore to a magnetic tape, in subfile format.
DOCUMENT	–	introduces a document on a basic peripheral.
FAIL	–	causes the object program to simulate a failed state.
FILEMLT	–	transcribes non-overlay binary programs from a magnetic tape to card files in the filestore.
HALT	–	puts the object program into a halted state.
LISTDIR	–	produces a listing of selected information about file entries contained in a directory.
LOADENTER	–	combines the functions of LOAD and ENTER.
MACDEF	–	enables the user to store a set of frequently used commands that can be implemented by a single command with run-time values given as parameters.
OFF	–	switches off specified bits in the switch word.
ON	–	switches on specified bits in the switch word.
POSTMORTEM	–	writes one or more regions of the core image of a saved or loadable file to the monitoring file system.
REALTIME	–	informs the system that the requirements of the currently loaded program are such that it must be kept permanently in store and plugged in.
TRAPCHECK	–	enables a user to check in what modes he is allowed to access a file or tape.
WAIT	–	suspends a job until a specified period of time has elapsed.

Appendix 2 Bibliography

This appendix contains a list of all relevant ICL 1900 Series manuals.

Operating systems GEORGE 3 and 4 (Edition 3) TP4169

GEORGE 3 Operation Management (Edition 1) TP4154

GEORGE 3 Operating (Edition 1) TP4140

7071 Teletypewriter Operating (Edition 1) TP4081

JEAN (Edition 2) TP4090

Algol: 16K Disc Compiler (Edition 1) TP4129

COBOL (Edition 1) TP4082

EMA manual (Edition 2) TP3146

FORTRAN: 32K Disc Compiler (Edition 1) TP4149

PLAN reference manual (Edition 1) TP4004

Library Specifications (Edition 2) TP4011

Index