

Oxford University Computing Laboratory

Computer Manuals

ICL 1900 Fortran 16k Disc Compiler

4300

SCIENTIFIC LANGUAGES

FORTRAN: 16K Disc Compiler.

1900

Library

MANUAL (NOTICE NO.)

27/9/72

4300

FORTRAN: 16K DISC COMPILER (1)

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

FORTRAN SEGMENTS IN NON-FORTRAN PROGRAMS

FORTRAN segments to be included in a program with a non-FORTRAN MASTER segment should be compiled at trace level 0. Segments compiled at trace level 1 may also be included, but trace 1 facilities will only be obtained if the segments include a READ or WRITE statement. The effect of including a FORTRAN segment compiled at trace level 2 in a program with a non-FORTRAN MASTER segment is not defined.

TEXT CONSTANTS

The maximum number of characters that can be specified in a text constant is 511.

OBJECT PROGRAM ERROR HALTS

In addition to the object program halt messages given on page 119, programs using disc files may halt with the message

HALTED:- SZ

This occurs if a disc logical bucket number is out of range or zero and can happen if an attempt is made to read from a scratch file before it has been written to.

Extension of Scratch Disc Files in FORTRAN programs

If a FORTRAN program opens a scratch disc file and further space is required, the FORTRAN library routines will attempt to make an appropriate extension. If there is no space available on the same cartridge as the scratch file, an extension onto any other suitable on-line cartridge will be made if the program is not run under UDAS. If UDAS executive is used, or if the scratch file is an online exofile under GEORGE 3, then an extension will not be made to another cartridge.

In either case, if more space cannot be allocated, the program will halt

NEED ED SCRATCH SPACE

or, if the scratch file is on fixed disc not under UDAS

NEED FD SCRATCH SPACE

If this occurs when UDAS is not used, the program may be restarted once a further cartridge with space available has been put online. Under UDAS executive (or GEORGE 3), the run must be repeated with a disc containing more spare space as the first online.

ARRAY ELEMENT BOUND CHECK

The following information amplifies that given on page 37 concerning the action at trace level 2 when an array element falls outside the bounds of the array.

For an array A, declared as $A(D_1, D_2, \dots, D_n)$, element $A(d_1, d_2, \dots, d_n)$ is held in the position (relative to the beginning of the array) specified by the formula:

$$d_1 + D_1(d_2 - 1) + D_1 D_2(d_3 - 1) + \dots + D_1 D_2 \dots D_{n-1}(d_n - 1)$$

The trace 2 check only verifies that an element is within the total area allotted to the array by checking that the position calculated from the above formula lies between 1 and $D_1 D_2 \dots D_n$ inclusive. Thus, in the example given on page 37 of an array declared as $\text{ARRAY}(3,2)$, a reference to $\text{ARRAY}(4,1)$ is not faulted. The position calculated by the formula above is $4 + 3 \times (1 - 1)$ or 4, which is less than 3×2 or 6, the array size.

In some cases, an array element reference where the product of the subscript expressions exceeds the size of the array is not faulted. For example, if an array is declared as:

DIMENSION B(3,4)

a reference to $B(5,3)$ will not be faulted: this element is considered to be held in position 11, within the bounds of the array.



L
RECEIVED 10 AUG 1973

PUBLICATION (NOTICE NO.)

1/8/73

4300

FORTRAN: 16K DISC COMPILER (2)

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

FORTRAN COMPILERS #XFAM AND #XFAE

The following additional execution errors can occur in programs compiled by #XFAM and #XFAE (all the errors are fatal):

<i>Code</i>	<i>Meaning</i>
109	Error in transfer on card input channel
110	Error in transfer on paper tape input channel
111	Error in transfer on magnetic tape input
112	Error in transfer on disc (EDS or FDS) input

FORTRAN COMPILERS #XFAM, #XFAE AND #XFAT

The following restrictions apply to use of the EQUIVALENCE statement in programs compiled by the above compilers.

- 1 A variable that is assigned a value in a DATA statement cannot be equivalenced to an array element.
- 2 An array may not be equivalenced to a shorter, non-common array that is wholly or partially initialized in a DATA statement.
- 3 An item appearing in a DATA statement may not be equivalenced to another item that appears in a DATA statement. An array or array element is considered to appear in a DATA statement if any element of the array is initialised in a DATA statement.

FORTTRAN COMPILERS #XFAM, #XFAE, #XFAT, #XFEW, #XFEV,
#XFIV, #XFEH AND #XFIH.

Formatted input records in FORTRAN object programs

The following rule should be observed when inputting formatted records:

If more than one sign appears before a number on the input record, the sign of the number is taken as that specified by the final plus or minus sign (that is, the one at the right hand end of the + - string) Hence + - 5 will have the internal representation of - 5, and - + 5 will have the internal representation of + 5. The additional signs will not be flagged as incorrect characters.

© International Computers Limited, Reading 1973



Library

PUBLICATION (NOTICE NO.)

7/1/76

4300 X
4159

FORTRAN: 16K DISC COMPILER (3) X
FORTRAN: MAGNETIC TAPE COMPILER (11)

File one copy of this
notice with each of the
publications indicated

FORTRAN COMPILER #XFAE

This notice announces the issue of a new version of the compiler #XFAE, which is intended for use with versions of the library SRF7 from Mark 23 onwards. The compiler has the mark number 5A and requires a minimum core size of 10624 words.

Errors corrected

The new issue corrects the errors reported in Scientific Languages Software Notices:

153/340, 156/343 and 209/2

In addition, a number of minor errors have been corrected.

DIRECT ACCESS FACILITIES

The following paragraphs describe new facilities which are now available in FORTRAN programs compiled by #XFAE/5A for the direct access of files on disc. These facilities are compatible with those already available in other 1900 compilers.

General information

Files may be processed serially or by direct access. In the former case the records are handled in the order in which they occur on the file, while in the latter case operations may be performed on selected records in any order.

Files specified to be processed by direct access (see 'The DEFINE FILE statement' below) may not be processed serially, and vice versa.

DIRECT ACCESS CHANNEL DESCRIPTION STATEMENTS

The format for a channel description statement specifying a direct access disc file is as follows:

$hk=xn/DIRECT (f (g))/b$

where

h is one of CREATE, INPUT, OUTPUT or USE, which are interpreted as follows:

- INPUT - use the named disc file which has been opened and written to in a previous program as a direct access disc file for input only. (See section Programming Techniques, item 3, page 5)
- OUTPUT - use the named disc file (assumed to contain no useful information) for output only.
- CREATE - use the named disc file (assumed to contain no useful information) for output followed by input if required.

USE - 1 with a permanent file: use the named disc file which has been opened and written to in a previous program as a direct access disc file for input and/or output. (See section Programming Techniques, item 3, page 5)

2 with a scratch file: use a scratch file for input and/or output.

k is the channel number or list of channel numbers by which the file is referenced in the FORTRAN program.

x is either ED or DA

n is the logical unit number for the disc file

f is either the file name or a dummy file name (see 'File Names' below).

g is the generation number (optional)
if (*g*) is omitted, the file of name *f* of highest generation number will be used.

b is the size in words of the buffer to be associated with the file (optional: see 'Buffer Size' below).

File names

f is either the name of a file or a dummy file name which will be replaced at run time by the actual file name (see the ICL 1900 Series manual *FORTRAN: 16K Disc Compiler*, TP 4300 page 14 section 'File names at run time'). All files to be used must already have been allocated using the file allocator program XPJC, or created using the GEORGE 3 and 4 command CREATE.

If the option USE or CREATE was specified in the program description, *f* and its enclosing parentheses may be omitted. In this case a scratch file will be created and will cease to exist when the program is deleted.

Buffer size

For optimum use of both main store and space on the disc, the value of *b* should be the same as the bucket size on the file.

The bucket size is defined when the file is allocated and may be 128, 256, 512 or 1024 words. If main store is at a premium, a buffer size less than the bucket size may be used for output files: this will reduce the object program store size but will be wasteful of space on the disc. The larger the value of *b*, the more main store will be occupied and in general the more efficient will be the object program. *b* should not be greater than the bucket size as only the first *n* words in the buffer (where *n* is the bucket size) would be used. If */b* or *b* is omitted, or *b* is zero, a buffer size of at least 128 words is assumed.

Direct access records, both formatted and unformatted, are held one or more to a bucket and may be up to *b*-4 words long.

File extension

If the file specified as a direct access file is not large enough to hold all the records an attempt will be made to extend the file.

This extension will be permitted only on the cartridge on which the file lies. Such extensions will be reported on the monitor output channel when the file is released (as for serial files).

It should be remembered that when calculating the size of the file required, one word must be added to the record size, as given in the DEFINE FILE statement, for the word count. Also, an extra bucket must be added for the end-of-file bucket.

Examples

1 CREATE 3 = ED3/DIRECT (FILE ONE)

Channel number 3 in the program corresponds to the direct access file FILE ONE to be allocated as ED3

Channel 4 in the program corresponds to a direct access scratch file with logical unit number 1. 256 word buckets will be used

DIRECT ACCESS File usage

Whereas a READ or WRITE statement referencing a serial file will cause input or output of the next record in sequence following that input or output by the last READ or WRITE statement, for a direct access file, records within the file may be accessed in any order. Each record within the file has a unique number associated with it, the records being numbered from one, and a record identified by specifying this number either as a constant or as a variable. For the appropriate forms of the READ and WRITE statements for direct access files see 'Input and Output for Direct Access Files' below.

The DEFINE FILE Statement

The DEFINE FILE statement is a specification statement and must appear before any executable statement. It describes the characteristics of any direct access file to be used. In order to use the direct access I/O operations each file must be described at least once by a DEFINE FILE statement as well as in the program description.

At run time the first DEFINE FILE statement encountered for one particular channel will set up the specification for that direct access file. Any subsequent statements encountered for that same channel will be ignored.

The DEFINE FILE statement has the form:

```
DEFINE FILE  $k_1 (m_1, x_1, t_1, v_1), k_2 (m_2, x_2, t_2, v_2), \dots, k_n (m_n, x_n, t_n, v_n)$ 
```

where

- k is an INTEGER constant identifying a particular channel.
- m is an INTEGER constant giving the number of records in the direct access file associated with channel number k .
- x is an INTEGER constant giving the size of each record in the direct access file associated with channel number k . The units in which the size is measured depends on t .
- t is one of the following characters:
 - U indicating that the file will contain only unformatted records. The record size will be measured in elements of two words.
 - E indicating that the file will contain only formatted records. The record size will be measured in characters.
 - L indicating that the file may contain both formatted and unformatted records. The record size is measured in half words.
- v is an INTEGER variable, the associated variable (see below), which may not be a dummy argument of a subroutine or function segment, but may be an item in a common block.

In any master, function or subroutine segment, DEFINE FILE statements must occupy a position between statement function definitions and executable statements. See the manual *FORTTRAN* TP 4261 page 40 Section 'Statement ordering'.

Example

The statement:

```
DEFINE FILE 4(50,100,L,I), 6(10,50,E,J), 1(40,20,I,K)
```

defines three direct access files. The first file, referred to as channel 4, consists of 50 records, of length 100 half words, that are to be transferred either with or without format control, the associated variable being I. The second file, referred to as channel 6, consists of 10 records, of length 50 characters, that are to be transferred with format control, the associated variable being J. The third file referred to as channel 1, consists of 40 records, of length 20 elements each of 2 words, that are to be transferred without format control, the associated

variable being K.

Input and Output for Direct Access Files

The appropriate forms of the READ and WRITE statements for transfer of direct access records are:

```
READ (k'r,l) list
WRITE (k',r,l) list
```

where

- k* is an integer constant or an integer variable representing the channel number. *k* must be followed by an apostrophe.
- r* is an integer expression which may or may not contain the associated variable, representing the record number.
- l* is present for formatted, but not for unformatted, records. If present, it is either the label of a FORMAT statement or the name of an array containing the format information.
- 'list' is an I/O list which is optional for READ or formatted WRITE statements.

Execution of either the READ or WRITE statement transfers the items in the 'list', in order, to the direct access file associated with *k*, starting at record number *r* of the file. (See Section Programming Techniques item 1 on page 5 for actual number of records transferred for each READ or WRITE.)

The FIND Statement

FIND statements are used to find the next record to be used in direct access file while the present record is being processed. Thus, the amount of time taken by a subsequent READ statement to that record is reduced. The FIND statement has the form:

```
FIND (k'r)
```

where *k* is an INTEGER variable identifying a particular channel number; *k* must be followed by an apostrophe. *r* is an INTEGER expression giving the number of the record to be found.

The file associated with channel number *k* must have been defined in a DEFINE FILE statement.

The FIND statement will find the record *r* in the direct access file associated with channel number *k*. The record will not be made available until a READ statement referring to that record is encountered. No advantage is gained in using the FIND statement before a WRITE statement.

An example of the use of this statement follows:

```
DEFINE FILE 6(100,60,E,I)
...
READ (6'1,9)A,B,C
FIND (6'2)
9 FORMAT (3F12.8)
VAL=SQRT (A*A+B*B+C*C)
...
READ (6'2,9)A,B,C
```

If these statements were included in a program, the FIND statement will cause record 2 of the direct access file associated with channel number 6 to be found whilst the data from record 1 is being processed. The second READ statement then makes record 2 available.

The Associated Variable

At the start of execution of the program, the associated variable of each direct access file defined in a DEFINE FILE statement is assigned the value 1, corresponding to the first record of the file.

The execution of any READ, WRITE or FIND statement referencing the file causes the value of the associated variable to be reset. A READ or WRITE statement will cause the associated variable to be set to the number of the record after the last record accessed by the READ or WRITE statement.

A FIND statement sets the associated variable to the record number specified in the FIND statement.

The expression *r* in any of the statements:

```
READ (k'r...  
WRITE (k'r...  
FIND (k'r)
```

may include the associated variable. If for a particular file *k* each READ and WRITE statement specifies the associated variable as the expression *r* the records will be accessed serially. Thus a direct access file can be used as though it were a serial file.

If the current value of an associated variable is to be transferred between segments this must be done by specifying the associated variable as an item in a common block. Associated variables may not be used as dummy or actual arguments.

Auxiliary I/O Operations and I/O Subroutines

- 1 BACKSPACE, ENDFILE, and REWIND have no effect on a direct access file
- 2 The subroutines FILE, RENAME, ALLOT, RUNOUT and RELEASE are available as for serial disc files

PROGRAMMING TECHNIQUES

- 1 Number of records transferred.

If the records concerned are formatted, input and output are controlled by the input/output list and the format specification as described in the section 'Effect of FORMAT statements and arrays' on page 43 of the ICL 1900 Series manual *FORTRAN*, Edition 2, 1971, TP 4261. The record lengths implied by the format specification must not be greater than the record size given in the DEFINE FILE statement. If the record size is greater than the length of a particular record as specified in a format specification for a WRITE statement, the record will be space filled.

If the records are unformatted, the number of records transferred is determined by the input/output list: transfer is complete when all items listed have had their values input or output, and as many records will be transferred as are required. A WRITE statement will cause any spare space in the last record to be zero filled

- 2 The DEFINE FILE statement for a direct access file does not have to be in the same segment in which the I/O operations occur. For example the DEFINE FILE statement can be given in the master segment with a subroutine performing the I/O operations. The associated variable will still be updated by the I/O operations even though it may not be stated explicitly in the subroutine
- 3 The following information is included to clarify the use of Execution Error 92 (see below):

When first accessed, any direct access file declared as OUTPUT or CREATE is opened and skeleton records, together with an end of file bucket, are written to the disc to correspond to the maximum number of records required.

When first accessed, any direct access file declared as INPUT or USE is opened and the first bucket is read. This is to ensure that the record word count which the first record should already contain corresponds to that calculated from the DEFINE FILE statement. Hence any direct access file declared in either of the program description statements INPUT or USE must already have been written to by a previous program. Execution Error 92 may be reported if the record word length of this file is not the same as the record word length specified in the DEFINE FILE statement

- 4 Example of direct access file handling

```
      DEFINE FILE 1 (1000,72,E,J)  
      DIMENSION A(100),B(100),C(100),D(100),E(100),F(100)  
      ...  
1     FORMAT (6F12.4)  
      FIND      (1'5)  
      ...  
      DO 2 I = 1,100  
      READ (1'J,1)A(I),B(I),C(I),D(I),E(I),F(I)  
      FIND (1'J+4)  
      ...  
4300(1)  
4159(11)
```

```

2    CONTINUE
    J = J-4
    DO 3 I = 1,100
3    WRITE (1'J+4,1)A(I),B(I),C(I),D(I),E(I),F(I)

```

This example illustrates the ability of direct access statements to read and write data in an order specified by the user. The first loop fills arrays A to F with data from the 5th, 10th, 15th ... 500th record of the file with channel number 1. Array A receives the first value in every record, B the second and so on, as specified by the FORMAT statement 1 and the input list of the READ statement. The first time the READ statement is executed, J is reset to 6; after the second execution, J is set to 11, and so on. At the conclusion of the first DO loop J has the value 505.

J is then reset to 501.

The second DO loop groups the same elements from each array as specified by the output list of the WRITE statement and the FORMAT statement 1. Each group of elements is written to the file with channel number 1, starting at the 505th record and continuing at intervals of five until record 1000 is written.

ERROR MESSAGES

Compilation Errors in Direct Access Input/Output Statements

Compilation error numbers have the same significance as in XFAE Mark 4D (see page 123 of the ICL 1900 Series manual *FORTTRAN: 16K Disc Compiler* Edition 2, 1971, TP 4300) with the following additions:

```

26  DEFINE FILE channel number is not small integer
27  DEFINE FILE Non-integer numbers of records or record size
    FIND      Apostrophe expected
28  DEFINE FILE Format specification not recognised
29  Direct Access READ  }
    Direct Access WRITE } Integer record number expected
    FIND                }
25  Amended to mean 'incorrect characters found'.

```

Execution Errors in Direct Access Input/Output Statements

Additions:

```

85  Channel number in DEFINE FILE statement does not refer to a direct
    access file.
86  File not large enough (even with extension) to hold all the records
    defined in the DEFINE FILE statement.
87  Record length in the DEFINE FILE statement larger than the minimum
    of the buffer and bucket sizes.
88  Relative position of a record is not a positive integer or exceeds
    the number of records in the data set.
89  Formatted record too long.
90  No previous DEFINE FILE statement.
91  Direct access file opened outside system.
92  DEFINE FILE statement does not correspond with previously created
    file.

```

Amendments:

```

23  An attempt has been made to read or write
    (1) serially to a direct access file
    (2) unformatted to a file described as E
    (3) formatted to a file described as U

```

Other errors and console halts as for serial I/O operations.

RESTRICTIONS PRESENT IN #XFAE

It should be noted that an item declared in a COMMON statement may not be equivalenced to another item which has also been declared in a COMMON statement.

The following restrictions apply to use of the DATA statement compiled by XFAE:

- 1 Hollerith strings of less than 5 characters assigned to a real variable, or less than 13 characters assigned to a double precision or complex variable, should be space-filled to at least 5 or 13 significant characters respectively to ensure that any remaining character positions allocated to the variable are space-filled by the compiler
- 2 The number of characters in a Hollerith string must not exceed the size specified by the variable, array or array element name to which it has been assigned

LIBRARY SUBGROUPSRF7

The following changes, which apply to the execution error messages, take effect with release of FORTRAN subroutine library routines RENAME/9, FILE/8 and %FINEDS/15.

- 1 Execution Error 28 has been redefined and the interpretation is now as follows:

An attempt has been made to use FILE or RENAME but a dummy file name has not been given in the Program Description, or

The new file name does not commence with an alpha character.
- 2 The message HALTED:NR may now occur once Execution Error 41, 42, 43 or 46 has been reported. This will signify that the program may not have released all its peripherals and hence any output from previous WRITE statements need not necessarily have been transferred to the appropriate media

ICL

FORTTRAN:
16K Disc
Compiler

1900 Series

RECEIVED 13 DEC 1971

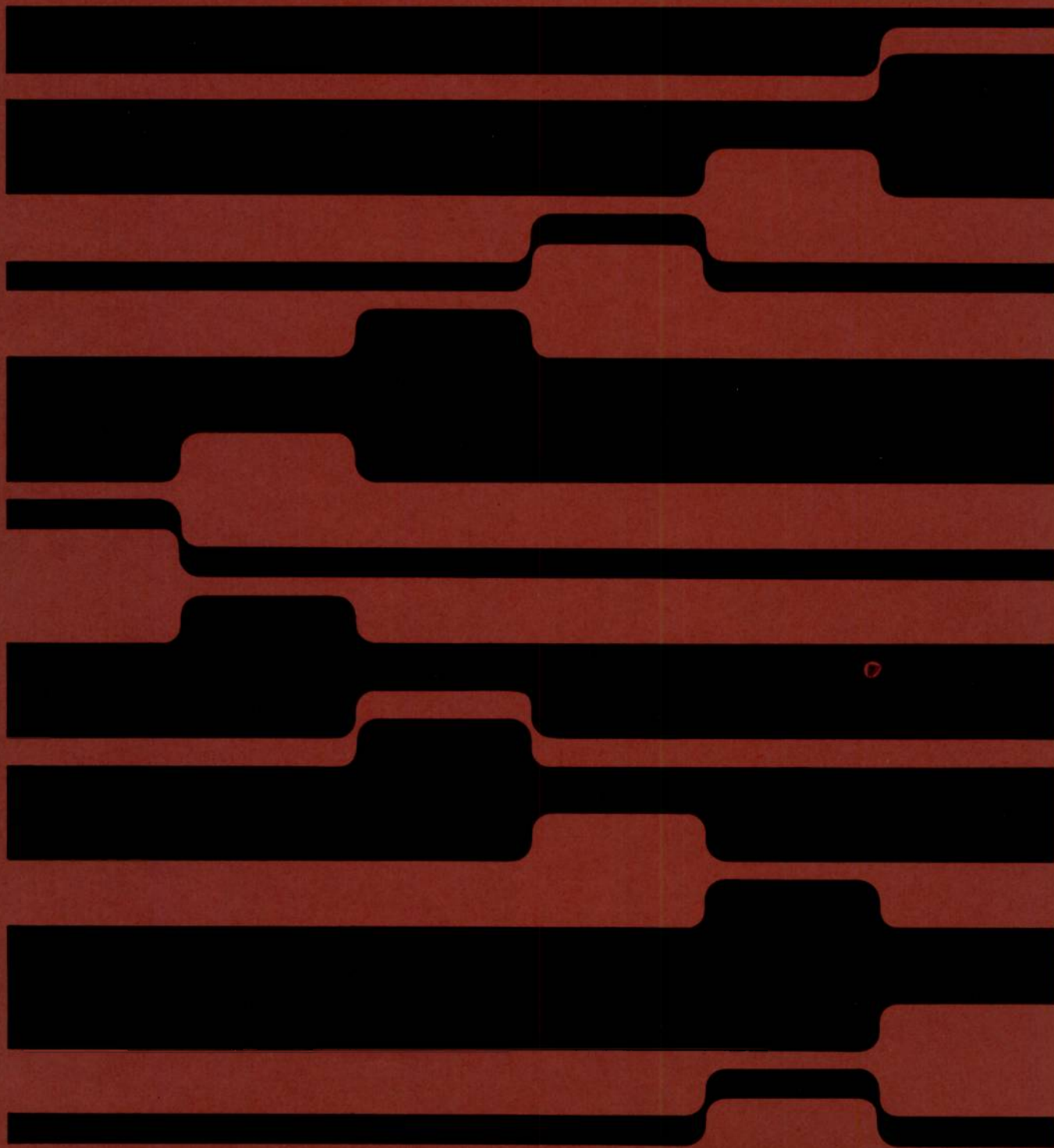
RECEIVED 15 DEC 1971

OXFORD UNIVERSITY COMPUTING LABORATORY

Copy 1.

COMPUTING SERVICE

4300



ICL

**FORTRAN:
16K Disc
Compiler**

1900 Series

OFFICE	COMPUTING LABORATORY
Copy 1.	PRINTING SERVICE 4300

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. Wherever practicable, ICL is willing to verify upon request the accuracy of any specific matter contained in this document.

Technical Publication 4300

© International Computers Limited 1969

First Edition February 1969

Reprinted April 1970
(Incorporating Amendment List 1)

Second Edition July 1971

Issued by Technical Publications Service
International Computers Limited
Head Office: ICL House, Putney, London SW15
and printed in Great Britain by
ICL Printing Services, Letchworth, Hertfordshire

Preface

OXFORD UNIVERSITY COMPUTING LABORATORY

Copy 1

COMPUTING SERVICE

4300

This manual describes the preparation of programs for compilation by the 16K disc version of the 1900 FORTRAN compiler and explains the operation of this compiling system. The system operates on programs written at the 1900 FORTRAN level (i.e. the additional facilities that 1900 Extended FORTRAN offers are not available for use with this compiler). Part 1 of the manual is intended for programmers who already have a knowledge of 1900 FORTRAN as described in the manual *FORTTRAN*. Part 2 is concerned with operating considerations.

Chapter 1 explains the use of compiling system statements which enable the programmer to control the course of a compilation and make the best use of the facilities provided by the compiler. The general use of disc and magnetic tape files is described in Chapter 2. The individual compiling statements are described in detail in Chapters 3 to 9. The use of the program designed to edit disc files of source program is described in Chapter 10. Chapter 11 describes how segments written in PLAN may be incorporated into FORTRAN programs and vice versa. Chapter 12 gives details of running the compiler under GEORGE 1 and 2. Chapter 13 is designed primarily for those responsible for the running of an installation and consists of a system description with particular reference to disc file organisation. Chapter 14 gives the operating instructions.

The minimum 1900 Series configuration recommended for using the FORTRAN 16K disc compiling system is as follows:

16K core store with at least 10,240 words unused by Executive or any operating system.

Floating point facilities (hardware or extracode).

Two E.D.S. transports, two Twin E.D.S. transports, or one F.D.S. transport.

One paper tape reader or one card reader.

One line printer.

Note: Throughout the manual wherever the term "disc" is mentioned, this should be taken to mean E.D.S. or F.D.S. or Twin E.D.S.

The following ICL 1900 Series manuals are referenced in this work:

Compiling Systems (Edition 1, TP4241)

FORTTRAN (Edition 2, TP4261)

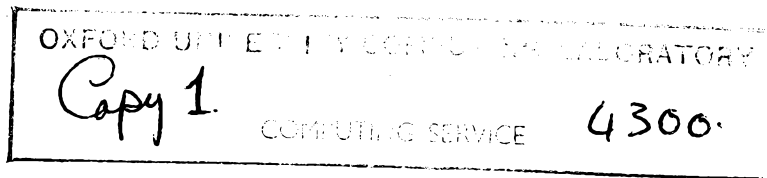
Direct Access Utilities (Edition 1, TP4190)

Magnetic Tape Utilities (Edition 1, TP4207)

FORTTRAN: Compiler Libraries (Edition 1, TP4170)

Operating Systems George 1 and 2 (Edition 3, TP4229)

Contents



Preface iii

Introduction xvii

PART 1 PROGRAMMING CONSIDERATIONS

Chapter 1 Program structure 1

COMPILING SYSTEM STATEMENTS 1

Initial statements 1

Program description statements 1

Intersegment statements 2

Terminator 2

Format of compiling system statements 2

Input sequence 3

Chapter 2 Disc and magnetic tape files 5

DISC FILES 5

Simple files, composite files and subfiles 5

Named files and scratch files 6

File and subfile names 6

Generation number 6

Retention period 6

Integrity code 7

FORTTRAN disc file formats 7

MAGNETIC TAPE FILES 7

Simple files, composite files and subfiles 7

Named files and scratch files 7

File and subfile names 7

Generation number 7

Retention period 8

Write permit ring 8

FORTTRAN magnetic tape formats 8

Chapter 3 Object program input/output 9

CHANNEL DESCRIPTION STATEMENTS 9

BASIC PERIPHERALS 10

Double buffering 10

DISC FILES 10

Effect of source program statements for disc files 10

READ AND WRITE STATEMENTS 11

ENDFILE STATEMENT 11

REWIND STATEMENT	11
BACKSPACE STATEMENT	11
Opening of disc files	11
Disc channel description statements	11
RECORD FORMAT	12
FILE NAMES	12
BUFFER SIZE	12
MAGNETIC TAPE FILES	12
Effect of source program statements for magnetic tape files	12
READ AND WRITE STATEMENTS	12
ENDFILE STATEMENT	13
REWIND STATEMENT	13
BACKSPACE STATEMENT	13
Opening of magnetic tape files	13
Magnetic tape channel description statements	13
RECORD FORMAT	14
BLOCK SIZE	14
FILE NAMES AT RUN TIME	14
FILE RENAMING	15
END OF FILE MARKERS IN OBJECT PROGRAM	
INPUT DATA	15
DUMP AND RESTART FACILITIES	15
Outline	15
Specifying the dump file	16
PRINTING OF FORMATTED DISC AND MAGNETIC TAPE FILES	16
CHANNEL 0	16
CONSOLE TYPEWRITER	16
CHARACTER TRANSFERS IN CORE STORE	17
SUPPRESSION OF TRANSFERS	18
 Chapter 4 Listing	 19
COMPILER LISTING	19
Full listing	19
Short listing	19
INITIAL LISTING STATEMENTS	19
INTERSEGMENT LISTING STATEMENTS	19
CONSOLIDATOR LISTING	20
 Chapter 5 Compiler output	 21
THE COMPILATION/CONSOLIDATION PROCESS	21
THE SEND TO STATEMENT	21
General	21
Normal usage	22
USE OF THE ICCLA-DEFAULT FILE	22
USE OF OTHER FILES	22
SYNTAX CHECKS	22
OUTPUT OF OBJECT PROGRAM	22
THE WORK STATEMENT	22
THE DUMP ON STATEMENT	23

THE RUN STATEMENT	23
SEPARATE CONSOLIDATION	23
SOME EXAMPLES OF THE COMPILATION/CONSOLIDATION PROCESS	24
 Chapter 6 Compiler input	 25
COMPILER INPUT MEDIA	25
Initial input medium	25
COMPILING FROM BASIC PERIPHERALS	25
COMPILING FROM DISC	26
The READ FROM statement for disc	26
Compiling system statements on disc	27
Editing of source programs on disc	27
COMPILING FROM MAGNETIC TAPE	27
Input from subfiles	27
Compiling system statements on magnetic tape	28
Input from simple files	29
Editing of source programs on magnetic tape	29
SEMI-COMPILED AND LIBRARY SUBFILES	29
The SEMICOMPILED statement	29
The LIBRARY statement	30
Use of the LIBRARY and SEMICOMPILED statements	30
 Chapter 7 Program testing	 33
COMPILATION ERROR DIAGNOSTICS	33
Error messages	33
LABEL ERRORS	33
MISSING SEGMENTS	33
OBJECT PROGRAM ERROR DIAGNOSTICS	34
Trace output channel	34
Trace level 1	34
NORMAL ERROR ACTION	34
OVERFLOW	34
User's check	35
THE ERROR TRAP FACILITY	35
Trace level 2	36
ERROR ACTION	36
STATEMENT TRACE output	36
ARRAY ELEMENTS	37
TRACE 2 WITH STEERING LISTS	37
Format of trace steering lists	38
TRACING EVERY STATEMENT	38
Suppression of trace output	39
Trace level 0	39
Mixed error detection levels	39
Operator intervention	39
User defined errors	39
Example of error detection in a program with an arithmetic error	39

Example of error trap facility	41
Chapter 8 Overlays	43
THE OVERLAY SYSTEM	43
PROGRAM DESCRIPTION STATEMENTS FOR OVERLAY PROGRAMS	43
The OVERLAY statement	43
The OVERLAY PROGRAM and OVERLAY SEGMENTS statements	44
The DEPTH OF OVERLAY statement	44
Restrictions	45
COMPILATION	45
OBJECT PROGRAM OPTIONALLY OVERLAID FROM MAGNETIC TAPE	45
RUN TIME EFFICIENCY	45
STANDARD FUNCTIONS IN OVERLAYS	45
EXAMPLE OF AN OVERLAY PROGRAM	45
Chapter 9 Miscellaneous features	47
EXTENDED DATA AND COMPACT DATA	47
Compiling and running on different size processors	47
Mixed language programs	47
PRIORITY	48
THE OMIT STATEMENT	48
THE COMPRESS STATEMENTS	48
Integer and logical statement	48
Double precision	49
Chapter 10 Editing	51
DESCRIPTION	51
General	51
Input	51
Output	51
ERROR DETECTION	51
LISTING OF A PROGRAM	51
DIRECTIVES	52
Batch directives	52
*IN	52
*OUT	53
*LOAD	53
The batch terminator	53
The subfile directive	53
The subfile terminator	53
Job directives	53
*NOLIST	53
*LIST	54
*CPCOPY	54
*TPCOPY	54
*DELETE	54

*LAST	54
Other directives	54
*ATLAS, *1900	54
*SWITCH	54
*WARN	54
*ALTER	55
Deletion and Amendment of lines	55
Insertion of lines	55
Order of *ALTER directives	55
Examples of input to #XMED	55
 Chapter 11 Mixed language programming	 57
COMMUNICATION OF DATA	57
TRANSFER OF CONTROL	57
CALLING PLAN SEGMENTS FROM FORTRAN	57
The FPROLOG subroutine	58
FUNCTION	58
DESCRIPTION	58
INPUT ARGUMENTS	58
RESULTS	58
USE OF OVERFLOW	59
SPECIAL POINTS	59
The FEPILOG subroutine	59
FUNCTION	59
DESCRIPTION	59
INPUT ARGUMENTS	59
RESULTS	59
USE OF OVERFLOW	59
Example	59
Arrays	60
THE GETAH SUBROUTINE	60
Function	60
Input arguments	60
Results	61
Notes	61
THE GENAH SUBROUTINE	61
Function	61
Input arguments	61
Results	61
EXAMPLE	61
Common storage areas	64
EXAMPLE OF THE USE OF COMMON	64
BLOCK DATA SEGMENTS	65
BLANK COMMON AREAS	65
CALLING FORTRAN SEGMENTS FROM PLAN	67
MIXED PLAN/FORTRAN OVERLAID PROGRAMS	69
The subroutine FOVER	69
FUNCTION	69
DESCRIPTION	69

INPUT ARGUMENTS	69
USE OF OVERFLOW	69
RETURN LOCATIONS	70
Notes	70
Restrictions	70
ENTRY POINTS AND MASTER SEGMENTS	71
USE OF WORD 30	71
PERIPHERAL USAGE	71
SUBPROGRAMMING	71
COMPILING MIXED PLAN/FORTRAN PROGRAMS	71
PLAN segments incorporated in a FORTRAN program	71
FORTRAN segments incorporated in a PLAN program	72
 Chapter 12 GEORGE 1 and 2	 73
INTRODUCTION	73
 STANDARD MACROS	 74
Macros available	74
Programming information	74
PARAMETERS	74
FORTRAN	77
FORTRANCOMP	79
FORTRANCOMPX	81
FORTRANRUN	83
FORTRANSEMI	85
 BATCH DEVELOPMENT SYSTEM	 87
General description	87
Operation of the System	87
Programming information	88
Tracing	88
DATA HANDLING	88
Macro specifications	88
STANDARD DEFAULT PROGRAM DESCRIPTION	89
RESTRICTIONS	89
PARAMETERS	89
AMENDBATCH	91
AMENDBATCHMT	93
FORTBATCH	95
FORTBATCHFD	95
NEWBATCH	97
NEWBATCHMT	99
Examples of use of the Batch Development System	101
 USER JOB DESCRIPTIONS	 105
Examples	105

PART 2 OPERATING CONSIDERATIONS

Chapter 13 System description	109
GENERAL DESCRIPTION	109
Compilation and consolidation	109
OBJECT PROGRAM	109
DISC FILE STRUCTURE	109
SYSTEM INITIALIZATION	109
File allocation	110
Obligatory disc files	110
COMPILER FILE	110
LIBRARY FILE	110
SEMI-COMPILED OUTPUT FILE	111
WORK FILE	111
Optional disc files	111
PROGRAM DUMP FILE	111
Efficiency considerations with E.D.S.	111
USE OF GEORGE	112
Batch Development System	112
Chapter 14 Operator's guide to the FORTRAN disc compiler #XFAE	113
HARDWARE REQUIREMENTS	113
Compiler and consolidator	113
Editor	113
USE OF PERIPHERALS	113
Compiler #XFAE	113
Consolidator #XPCK (Mk 11C and later)	114
Editor #XMED	114
Object program	114
PRIORITY	114
OPERATING INSTRUCTIONS	115
Compilation and consolidation	115
Loading and running the object program	115
Loading and running the editor, #XMED	116
DISC ERROR CONDITIONS	117
OTHER EXCEPTION CONDITIONS	117
Compilation	117
Consolidation	119
Loading and running the object program	119
Loading and running the editor, #XMED	120
RUNNING #XFAE UNDER GEORGE	121
Batch Development System	121
Appendix 1 Compilation/Consolidation errors	123
COMPILATION ERROR NUMBERS	123
CONSOLIDATION ERROR MESSAGES	125

Appendix 2 Execution error numbers	127
Appendix 3 Paper tape and card codes	131
Appendix 4 Data formats on disc files	133
Appendix 5 Data formats on magnetic tape files	135

Illustrations

Figure 1	Multi-level subfile structure	5
Figure 2	Full listing without errors	20

Tables

Table 1	Compiling system statements	3
Table 2	Statements in a STATEMENT TRACE list	37
Table 3	Location of results for different functions	57
Table 4	The method of storage of FORTRAN variables	66
Table 5	Value of operand in PLAN instructions when calling a FORTRAN segment from a PLAN segment	67

Introduction

1900 SERIES COMPUTERS

Computers in the 1900 Series differ greatly in size, power and the number and type of peripheral units. The smaller machines run only one program at a time and are controlled by a human operator. The larger machines are multiprogramming; that is, they can run several programs simultaneously and may be controlled by one of the various operating systems available for different configurations.

Whether multiprogramming or not, each machine has supplied with it a program called *Executive*. This program may be regarded as a permanent part of the computer and works in conjunction with the operating system or human operator. It controls the multiprogramming activities of the larger machines: it also controls the peripherals and executes requests for the transfer of information between peripherals and the central processor. It also communicates with the operator and executes orders given by the operator. This communication is achieved via the *console typewriter*, which gives a permanent typed record of all communications in the order in which they take place. The operator can type instructions to Executive in order to carry out certain functions, for example to load, activate and delete programs. Executive also can use the typewriter in order to inform the operator of any unusual situations within the machine.

1900 SERIES COMPILERS

Compilers written for use with 1900 Series computers employ a common system of compiling. In this system a compiler does not translate source programs directly into machine code form; instead, programs written in any source language are first converted, by the appropriate compiler, into a series of segments in a common, intermediate language called *semi-compiled*. To produce machine code form the semi-compiled segments are combined by a routine known as the *consolidator*. For machines using discs, the output from the consolidator consists of a binary object program either loaded into core store ready for running or in a disc file in a loadable form.

The great advantage of this system is that semi-compiled segments derived from more than one source language may be combined together at the consolidation stage.

THE FORTRAN 16K DISC SYSTEM

The 1900 Series FORTRAN 16K disc system consists of a compiler #XFAE automatically linked with a consolidator. A set of *compiling system statements*, resembling normal FORTRAN statements in format is used to supply the compiler with the information necessary to produce an object program. A program, or segments of a program, may be compiled from paper tape, cards, magnetic tape, or disc. The compiler outputs unconsolidated semi-compiled to disc. Consolidation and loading of the semi-compiled is carried out by the consolidator. The consolidation is normally an automatic process immediately following compilation, with the compiler passing information to the consolidator. However, if required, the consolidation phase can be carried out as a completely separate process.

The compiler is overlaid from either E.D.S. or F.D.S. If the overlay medium is E.D.S., the cartridge on which the compiler itself is held must be on-line throughout the compilation run. The compiler contains facilities for compiling an overlaid or a non-overlaid program. The compiler may be run under normal operator control or under the control of the GEORGE operating system.

During compilation, an optional listing may be output to a line printer.

PART 1 PROGRAMMING CONSIDERATIONS

Chapter 1 Program structure

COMPILING SYSTEM STATEMENTS

A FORTRAN program written according to the rules given in the 1900 Series FORTRAN manual is incomplete in that it does not contain sufficient information to enable the compiler to produce an object program. Certain statements must be added to the basic program to provide the compiler with the necessary information; these statements are known as compiling system statements. There are three types of compiling system statements: *initial statements*, *program description statements* and *intersegment statements*. Initial statements and intersegment statements provide the compiler with steering information. Program description statements enable the object program to be completely defined.

A complete program to be compiled consists of the following parts, in the order below.

- 1 Initial statements.
- 2 Program description statements.
- 3 Source or semi-compiled segments with, optionally, intersegment statements between the segments.
- 4 A FINISH statement.

Table 1 on page 3 lists all the compiling system statements.

Initial statements

Initial statements specify the files that are to be used for the compiler output and the type of listing required. In addition, there are initial statements for scanning semi-compiled library subfiles if it is required to incorporate into a FORTRAN object program library routines that are not held in the compiler library. If a dump of the program is required on disc after consolidation, the DUMP ON statement must be present.

Initial statements may be input in any order before the program description statements. The DUMP ON statement must be present if an overlaid program is to be consolidated automatically following compilation; all other initial statements are optional.

Program description statements

The program description gives information about the object program to the FORTRAN disc compiling system. Program description statements associate source program channel numbers with 1900 system names, specify the level of error tracing that is to be incorporated in the object program, define the priority of the program and so on. The program description statements taken together constitute a segment (which is given the special name %%%F) and this segment must be the first segment input to the compiler.

A program description must begin with one of the following statements:

```
PROGRAM (name)
OVERLAY PROGRAM (name)
SEGMENTS (name)
OVERLAY SEGMENTS (name)
```

name is a four character program name which will be used to refer to the program when it is loaded and run.

The **PROGRAM** statement is used to introduce a non-overlaid program which is to be compiled and consolidated.

The **OVERLAY PROGRAM** statement is used to introduce an overlay program which is to be compiled and consolidated (see Chapter 8, page 43).

When a source program is not complete, or if consolidated object program is not required, the program description must be introduced by a **SEGMENTS** statement.

The **OVERLAY SEGMENTS** statement is used instead of a **SEGMENTS** statement in order to introduce segments to be compiled which may later be included in an overlay program (see Chapter 8, page 43).

The last statement in the program description must be the statement

END

Other program description statements are optional and may appear in any order between the first and last statements of the program description.

Intersegment statements

Certain initial and program description statements may be input between source segments if it is required to alter the original specifications during compilation.

For example, the input medium may be changed between segments by the **READ FROM** statement.

Terminator

The end of the input to the compiler must be indicated by the terminator statement:

FINISH

Format of compiling system statements

All compiling system statements resemble normal **FORTRAN** statements and are written in columns 7 to 72 on normal coding sheets. Spaces are ignored except in file and subfile names. No continuation lines are permitted within compiling system statements.

Table 1 on page 3 lists all the compiling system statements.

<i>Group</i>	<i>Statement</i>	<i>See note</i>
Initial statements	LIST/SHORTLIST/NO LIST	1
	MAP	1, 4
	SEND TO	1, 4
	DUMP ON	1, 4
	RUN	1, 4
	READ FROM	1, 3, 4
	WORK	1, 4
	LIBRARY	1, 3, 4
	SEMICOMPILED	1, 3, 4
Program description statements	PROGRAM/OVERLAY PROGRAM/SEGMENTS/OVERLAY SEGMENTS	2
	COMPACT DATA/EXTENDED DATA/MIXED SEGMENTS	1, 4
	COMPRESS INTEGER AND LOGICAL	1, 4
	COMPRESS DOUBLE PRECISION	1, 4
	INPUT	1, 3, 4
	OUTPUT	1, 3, 4
	USE	1, 3, 4
	CREATE	1, 3, 4
	OMIT	1, 3, 4
	OVERLAY	1, 3, 4
	DEPTH OF OVERLAY	1, 4
	TRACE	1, 4
	PRIORITY	1, 4
	END	2
Intersegment statements	READ FROM	1, 3, 4
	LIST/SHORTLIST	1, 3, 4
	TRACE	1, 3, 4
Terminator	FINISH	2

Table 1 Compiling system statements

Notes:

- 1 Optional statement.
- 2 Obligatory statement.
- 3 More than one such statement may occur within the group.
- 4 Order not essential within the group.

Input sequence

The sequence of the input for any compilation run is as follows:

Initial statements

PROGRAM/OVERLAY PROGRAM/SEGMENTS/OVERLAY SEGMENTS

.....
 *Program description statements

.....
 END

Source segments

.....
 *Intersegment statements

.....
 Source segments

.....
 FINISH

The sections indicated by an asterisk * are optional.

Example

The following example illustrates a possible combination of compiling system statements with their effects. The compiling statements chosen will cause the program to be automatically consolidated and loaded into core store ready for running. This is the straightforward use of the compiler but many other variations are possible and these are described in later chapters.

Program description statements	LIST SEND TO (PROGRAMVFILE, SUBFI) PROGRAM (TEST) INPUT 1 = CR0 OUTPUT 2 = LPO TRACE 2 END	Full listing on a line printer required. Unconsolidated semi-compiled program to be sent to a subfile SUBFI on a file called PROGRAMVFILE Object program called TEST Source program channel 1 is identified as card reader 0. Source program channel 2 is identified as line printer 0. Tracing at level 2 will be incorporated in the object program.
	Segments	Segments written according to the 1900 Series manual <i>FORTRAN</i>.
	FINISH	

Chapter 2 Disc and magnetic tape files

This chapter describes the structure of disc and magnetic tape files used in the FORTRAN disc system.

DISC FILES

Information is stored on disc in *files*. The files are set up on disc by using one of the file allocator programs #XJEC or #XJFC. (The specifications for these programs are given in the manual *Direct Access Utilities*.)

Simple files, composite files and subfiles

There are two types of files on discs: *simple* files and *composite* files. Simple files can hold only one category or group of information; composite files can hold many categories or groups of information, possibly quite unrelated to each other and of different types (e.g. source program and data).

A simple file on disc can contain only data.

A composite file on disc can contain the following categories of information

- 1 Binary program
- 2 Semi-compiled segments
- 3 FORTRAN source program

In 1900 FORTRAN disc usage, data required by a FORTRAN object program is always held on disc in simple files.

For composite files, each separate category or group of information is held in a subdivision of the file known as a *subfile*. Thus a composite file consists of one or more subfiles. A simple file is defined as a file with no subfile structure. Subfiles can themselves be composite i.e. they can contain one or more other subfiles; this is known as a *nested* subfile structure. If a subfile does not contain another subfile it is known as a *simple* subfile.

In the most elementary composite file structure, the file consists of one subfile containing useful information, e.g. source program, and a *free space* subfile which consists of the remaining unused part of the disc file. New subfiles, containing additional information can be added to a composite subfile. When this occurs, a new subfile is formed from part of the free space subfile. Unlike other subfiles, the free space subfile has no name. (See the section *File and subfile names*.)

Composite files containing only simple subfiles are said to possess a *single level* subfile structure. A file containing composite subfiles is said to have a *multi-level* subfile structure. The composite subfiles making up the file are known as *top level* or *level 0* subfiles. The subfiles contained in these composite subfiles are known as *level 1* subfiles. Further, these subfiles may also be composite and contain *level 2* subfiles and so on.

Figure 1 illustrates a multi-level subfile structure.

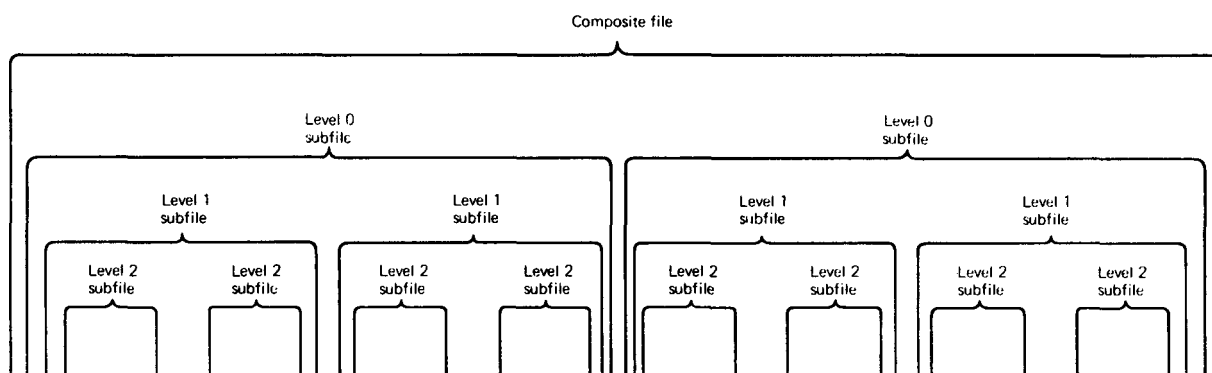


Figure 1 Multi-level subfile structure

Subfiles at a given level need not physically follow one another on disc, but often will. In 1900 FORTRAN usage of disc files, subfile structure will normally be single level with one named subfile containing information and a free space subfile.

Named files and scratch files

A disc can contain a number of named files. *Named files* can be used to store information which is to be retained. *Scratch files* are used as work space during the running of a program. They have no names and exist only as long as required by the program. In the FORTRAN disc system, work space is required during the compilation/consolidation process and a scratch file is used unless a WORK statement is specified. (See *The work statement*, page 22). Scratch files may also be used by the programmer in object programs. A scratch file can be set up only if free space, i.e. space not allocated to permanent files, is available on the same disc. Multi-programming installations may prohibit the use of scratch files.

Permanent files are created on E.D.S. or F.D.S. by using one of the file allocator programs #XJEC or #XJFC. E.D.S. files should be allocated on one cartridge only. The file allocator may also be used to remove files from disc.

The size of the files and the specific areas of the disc used are defined on allocation. However, both permanent output files and scratch files will be extended by the compiler or object program, if required. For permanent files the extension will be on the disc containing the current end of file. The size of the extension depends on both the file size before extension and the space available on the disc.

The basic unit of information in disc systems is the *bucket*. A file consists of a number of buckets, this number being defined on allocation. The bucket size for data files may be 128, 256, 512 or 1024 words; for all other files it must be 128 words. The size is defined on allocation. All buckets in the file are the same size.

File and subfile-names

Each file and subfile on disc is identified by a name consisting of up to 12 characters chosen from the following set:

A to Z

0 to 9

Space

Hyphen (-)

The first character *must* be a letter.

It is possible for two subfiles in the same file to have the same name. In the manual, file names are denoted by *f* and subfile names by *s*.

Generation number

All disc files used by an installation should be uniquely identified. Files that contain different versions of the same information may be distinguished by the addition of an integer *generation number* written in parentheses after the file name. e.g. MYSOURCEFILE (10). File and subfile names with generation numbers are referred to in this manual as follows:

$f(g)$, $s(g)$

In any context in which these forms are allowed, the simple forms *f* and *s* are also allowed.

No check on file and subfile generation numbers is made if the generation numbers are not specified. In the case of a file, the file of required name with highest generation number is selected if the generation number is not specified; in the case of a subfile, the top level subfile of required name with highest generation number is selected.

Retention period

To prevent the contents of disc files being inadvertently overwritten it is possible to specify the length of time (in days) that the contents of the file will be protected i.e. not overwritten, counting from the day information was last written to the file. When this *retention period* has expired (or expressing it another way the *purge date* has been exceeded) the information on the file can be overwritten by programs which explicitly request this file in the normal way. The new information then written to the file will likewise be protected for the requisite number of days. When the file is originally allocated by #XJEC or #XJFC, it is given a retention period of zero. Certain utility programs outside the FORTRAN system can give a file a retention period or change its value. However, it must be stressed that in FORTRAN disc usage neither the FORTRAN disc compiler nor compiled programs can change the retention period.

In FORTRAN disc usage, it is acceptable for most files to have zero retention periods. For very important files that are not to be changed a utility program may be used to set a retention period.

Integrity code

In a multiprogramming environment, it is possible for more than one program to access a given file at any one time. The *integrity code* is used to control this procedure. A file can have three different codes associated with it, the codes being defined when the file is allocated. The three codes have the following meanings:

- 1 Only one program may access a file at any one time.
- 2 Any number of programs can access a file at any one time, but no more than one program can *write* to a file at any one time.
- 3 Any number of programs can access a file simultaneously unless there is a program writing to it, in which case no other program can access the file.

Code 2 is suitable for most FORTRAN applications.

FORTRAN disc file formats

Details of the formats of data on FORTRAN disc files are given in Appendix 4, page 133.

This is not normally of interest to programmers writing solely in FORTRAN.

MAGNETIC TAPE FILES

Magnetic tape files may be used to hold FORTRAN source program for input to the compiler and for input and output of data in object programs.

Simple files, composite files and subfiles

Information is stored on magnetic tape in the form of a file. In the ICL 1900 Series FORTRAN magnetic tape system, a file occupies one reel of magnetic tape. A file may be a *simple file* containing one category of information, or it may be a *composite file* holding many categories of information, possibly quite unrelated to each other. A composite file contains one or more subfiles. In the simplest composite file structure one subfile follows another. Subfiles can themselves be composite i.e. they can contain one or more other subfiles; this is known as a *nested* subfile structure. In the FORTRAN system simple files are used only for holding data required by an object program whilst composite files are used only to hold source program to be input to the compiler.

Named files and scratch files

On most computer installations magnetic tapes are divided into two groups, *named files* and *scratch files*. Named files are used when the information is to be retained. Each named file is given a unique name and is assigned to a particular user or group of users; a named file can be opened only if it is requested by name. A scratch file can be used by any user as a work file during the running of a program; once the program has finished, the scratch file is not considered to contain any useful information and can be used by another program.

File and subfile names

Each file and subfile on magnetic tape is identified by a name consisting of up to 12 characters chosen from the following set:

- A to Z
- 0 to 9
- Space
- Hyphen (-)

The first character must be a letter.

In the manual file names are denoted by *f* and subfile names by *s*.

Generation number

All magnetic tape files used in an installation should be uniquely identified. Files that contain different versions of the same information may be distinguished by the addition of an integer generation number written in parentheses after the file name e.g. MYSOURCEFILE (10)

Magnetic tape file and subfile names and generation numbers are referred to in this manual as follows:

$$f(g), s(g_s)$$

The specified generation number is stated when a named file is created. When a named file is accessed, the generation number of the file will be checked against the specified generation number. If the generation number is specified as zero, or is omitted, no check will be made.

Retention period

All named files have a *retention period*. This is the length of time (in days) for which a file will be retained as a named file. It is also the number of days which the file will be protected against inadvertent overwriting. When the retention period has expired the file, by definition, becomes a scratch file. A scratch file has zero retention period and may have any name.

Under the FORTRAN magnetic system, it is possible to rename and re-use named files once the information on them is no longer needed. Therefore it is recommended that all named files have the highest possible retention period i.e. 4095 days. When a tape file is created in an object program by means of a CREATE statement (see *Magnetic tape channel description statements*, page 13) a retention period can be specified by the user. This retention period may be specified as an integer in the range 0 to 4095; the tape is given a retention period of 4095 if no retention period is specified.

Write permit ring

The *write permit ring* is a small plastic ring that must be fitted to a magnetic tape spool before the magnetic tape may be written to; the absence of a write permit ring thus prevents the accidental overwriting of information which is to be retained.

FORTRAN magnetic tape formats

Details of the formats of data on FORTRAN magnetic tape files are given in Appendix 5, page 135.

This is not normally of interest to programmers writing solely in FORTRAN.

Chapter 3 Object program input/output

CHANNEL DESCRIPTION STATEMENTS

Input/output statements within a FORTRAN source program refer to their related peripheral units by integer *channel numbers*. These must be in the range 1 to 4095 (0 may be used in special cases) in 1900 FORTRAN. When the program is compiled these channel numbers are associated with the 1900 system names of the peripheral units by *channel description statements* in the program description.

There are four types of channel description statements, as follows:

INPUT	$k = Z$ (Input only)
OUTPUT	$k = Z$ (Output only)
USE	$k = Z$ (Input and output)
CREATE	$k = Z$ (Input and output)

k can be a single source program channel number or else a string of such numbers separated by commas. In the first case Z gives details of the peripheral to be associated with k ; in the second case each channel is associated with the same peripheral Z . The same Z should not appear in more than one channel description statement. Also, k must be unique within any given program.

Any number of different channel numbers may be defined. Some peripherals can hold either formatted or unformatted records; others may hold only formatted records. Normally, at least one formatted output channel should be available and this must be defined in the first OUTPUT statement of the program description. This channel is used for the output of diagnostic information. (See Chapter 7, page 33.)

Channels can be defined in the program description even if they may not be used on particular runs of the program. However, the more channels that are defined (particularly if they are on different types of peripheral) the more space is used by the input/output system in the object program.

Part of Z is normally a 1900 *system name* of the relevant peripheral. This consists of a two letter code followed by a *logical unit number*, e.g. LP2 is a line printer with logical unit number 2. These numbers are chosen by the programmer and are then assigned to a particular peripheral or file by the 1900 system. Logical unit numbers must be in the range 0 to 15. Numbers for peripherals or files with the same code must be distinct. Thus, LP2 should not appear twice in the program description (unless on any particular run of the program it will be referred to by only one of the relevant channel numbers) but a card punch could be CP2.

The peripherals and input/output facilities available to the object program are:

- Paper tape reader
- Paper tape punch
- Card reader (1900 or ATLAS code)
- Card punch (1900 or ATLAS code)
- Line printer
- Magnetic tape
- Exchangeable disc store (serial files)
- Fixed disc store (serial files)
- Console typewriter
- Core store transfers
- Suppression of transfers

BASIC PERIPHERALS

Details of programming for basic peripherals appear in the 1900 Series FORTRAN manual. Only formatted records may be processed. Basic peripheral names may appear only in INPUT and OUTPUT statements with one of the following forms:

INPUT	$k = \text{TR}n$	Paper tape in 1900 code
INPUT	$k = \text{CR}n$	Cards in 1900 code
INPUT	$k = \text{CR}n/\text{ATLAS}$	Cards in ATLAS code
OUTPUT	$k = \text{TP}n$	Paper tape in 1900 code
OUTPUT	$k = \text{CP}n$	Cards in 1900 code
OUTPUT	$k = \text{CP}n/\text{ATLAS}$	Cards in ATLAS code
OUTPUT	$k = \text{LP}n$	Line printer
OUTPUT	$k = \text{LP}n/l$	Line printer

k is the channel ($1 \leq k \leq 4095$; exceptionally, $k = 0$)

n is the logical unit number ($0 \leq n \leq 15$)

l is the line printer line-length, i.e. 96, 120 or 160; absence of l implies 120 characters.

Examples

1 INPUT 3 = CR0

Source program channel number 3 is associated with the card reader assigned as CR0.

2 OUTPUT 4, 5, 6 = LP0/160

Source program channel numbers 4, 5 and 6 are associated with a 160 print position line printer assigned as LP0.

Double buffering

Transfers between basic peripherals and FORTRAN object programs are normally single buffered; double buffering may be specified for one basic peripheral of each type by assigning to it the logical unit number 7. There is one exception: punched card devices using ATLAS card code cannot be double buffered whatever the logical unit number assigned to them.

Double buffering increases the transfer rate of a peripheral to which it is applied and is therefore desirable in any program involving large numbers of transfers; this increased transfer rate is obtained at the cost of an increased core store requirement.

Examples

INPUT	2,6 = TR7
INPUT	8 = TR0
INPUT	3 = CR7
OUTPUT	4 = LP7
OUTPUT	5 = CP7/ATLAS

At run time, units TR7, CR7 and LP7 will be double buffered; TR0 and CP7 will be single buffered.

DISC FILES

Simple files held on disc may be processed serially using either formatted or unformatted READ and WRITE statements.

Effect of source program statements for disc files

The following sections describe the effects of various FORTRAN statements when applied to disc files. General programming information is given in the 1900 Series manual *FORTRAN*.

In the following sections, the channel number is denoted by c , meaning an integer variable or constant.

READ AND WRITE STATEMENTS

A READ statement inputs the next available record. A WRITE statement outputs a record after the last record that was read or written (unless the file has been re-positioned by using REWIND or BACKSPACE). A WRITE statement that attempts to overwrite an already written record will destroy that record and any following records on the file.

ENDFILE STATEMENT

Processing of an output file must be terminated by execution of an ENDFILE statement after the last record has been output by a WRITE statement. If no ENDFILE statement is executed the file may be left in a non-standard format and may not be readable.

The form of the statement is:

ENDFILE *c*

REWIND STATEMENT

The REWIND statement will logically re-position the specified file so that the next READ or WRITE statement applies to the first record of the file.

The form of the statement is:

REWIND *c*

BACKSPACE STATEMENT

The BACKSPACE statement logically re-positions the file so that the previous record is available.

The form of the statement is:

BACKSPACE *c*

Opening of disc files

Before a disc file is used it must be 'opened', i.e. made available by the 1900 system. In general, this happens automatically the first time a FORTRAN statement referring to the file is executed. If required, the user may open the file earlier by using the ALLOT subroutine. For details see the manual *FORTRAN: Compiler Libraries*.

Disc channel description statements

The format for a channel description statement specifying a disc file is as follows:

$$hk = x \ n/q(f(g))/b$$

h is one of INPUT, OUTPUT or USE.

k is the channel, as defined previously.

x is either ED or FD. ED specifies an E.D.S. file and FD specifies an F.D.S. file.

n is the logical unit number for the disc file.

q is the qualifier defining the format of records on the disc file (see below).

f is either the disc file name or a dummy file name (see below).

g is the file generation number (optional).

b specifies the size in words of the buffer in the program to be associated with the file (optional).

The different values of *h* are interpreted as follows:

INPUT	use the named disc file for input only.
OUTPUT	use the named disc file for output only.
USE	use the named disc file for input and/or output.

The CREATE statement is not used with disc files.

RECORD FORMAT

q, the qualifier defining the format of records on the disc file, must be one of the following:

FORMATTED for formatted records
UNFORMATTED for unformatted records

In special cases, *q* may take the form DUMP, see *Dump and restart facilities*, page 15.

FILE NAMES

f is the name of a file which must already have been allocated. Alternatively, *f* may be a dummy name, to be replaced at run time by the actual file name (see *File names at run time*, page 14).

The file name and its enclosing parentheses may be omitted for the USE statement. In this case a scratch file will be created. The scratch file will cease to exist when released or when the program is deleted.

BUFFER SIZE

For optimum use of both core store and space on the disc, the value of *b* should be the same as the bucket size on the file. This physical bucket size on the disc can have one of four values i.e. 128, 256, 512 or 1024 words, being defined when the file is allocated. For programs whose core store requirement is at a premium, a buffer size less than the bucket size may be used for output files as this will reduce the object program core size, although it will be wasteful of space on the disc. The larger the value of *b*, the more core store will be occupied and in general the more efficient will be the object program. *b* should not be given a value greater than the bucket size as only the first *n* words in the buffer (where *n* is the bucket size) would be used.

b and the preceding solidus/may be omitted, in which case a buffer size of at least 128 words is assumed.

For scratch files, the bucket size used will be the first permissible bucket size greater than the specified buffer size, unless the buffer size specified is greater than 1024 words in which case the bucket size will be 1024 words. If no buffer size has been specified the bucket size will be chosen arbitrarily.

Formatted records are held one or more to a bucket. Maximum record length is *b* - 3 words. Each unformatted record starts a new bucket and occupies as many further buckets as are necessary to hold the whole of the record. There is no maximum record length for unformatted records. Note that five words of each bucket are used to hold organisational information in unformatted files.

Examples

1 INPUT 1 = ED1/FORMATTED (EDITFILE)

Channel number 1 in the program corresponds to the formatted file EDITFILE to be allocated as ED1.

2 USE 4 = FD2/UNFORMATTED/1024

The compiler associates channel number 4 in the program with an unformatted F.D.S. scratch file with logical unit number 2. 1024 word buckets are to be used.

MAGNETIC TAPE FILES

Simple files held on magnetic tape may be processed using either formatted or unformatted READ and WRITE statements.

Effect of source program statements for magnetic tape files

The following sections describe the effects of various FORTRAN statements when applied to magnetic tape files. General programming information is given in the 1900 Series *FORTRAN* manual.

In the following sections, the channel number is denoted by *c*, meaning an integer variable or constant.

READ AND WRITE STATEMENTS

A READ statement inputs the next available record. A WRITE statement outputs a record after the last record that was read or written (unless the file has been re-positioned by using REWIND or BACKSPACE). A WRITE statement that attempts to overwrite an already written record will destroy that record and any following records on the file.

ENDFILE STATEMENT

Processing of an output file must be terminated by execution of an ENDFILE statement after the last record has been output by a WRITE statement. If no ENDFILE statement is executed the file may be left in a non-standard format and may not be readable.

The form of the statement is:

ENDFILE *c*

REWIND STATEMENT

The REWIND statement will rewind the specified file and position it so that the next READ or WRITE statement applies to the first record of the file.

The form of the statement is:

REWIND *c*

BACKSPACE STATEMENT

The BACKSPACE statement positions the file so that the previous record is available. For a formatted record, physical tape movement may or may not take place.

Repeated backspace operations lead to inefficient tape usage and should be avoided if possible. In some cases REWIND followed by repeated READ statements may be preferable.

The form of the statement is:

BACKSPACE *c*

Opening of magnetic tape files

Before a magnetic tape file is used it must be 'opened', i.e. made available by the 1900 system. In general, this happens automatically the first time a FORTRAN statement referring to the file is executed. If required, the user may open the file earlier by using the ALLOT subroutine.

Alternatively, the user may wish to open the file himself by means of a PLAN subroutine. This is acceptable to the FORTRAN system, which always checks if a file is already open before itself attempting to open it.

Magnetic tape channel description statements

The format for a channel description statement specifying magnetic tape is as follows:

$$hk = MTn/q(f(g,r))/b$$

h is one of INPUT, OUTPUT, USE, CREATE.

k is the channel, as defined previously.

n is a magnetic tape logical unit number.

q is the qualifier defining the format of the records (see below).

f is the file name. Alternatively, it may be a dummy file name, to be replaced at run time by the actual file name.

g is the file generation number (optional).

r is the retention period and is used only in CREATE statements.

b is the block size in words (optional—see below).

The different values of *h* are interpreted as follows:

INPUT	use the named magnetic tape file for input only. A write permit ring must not be present.
OUTPUT	use the named magnetic tape file for output only. A write permit ring must be present.
USE	use the named magnetic tape file for input and/or output. A write permit ring must be present.
CREATE	obtain a scratch tape, rename it, use it for output, followed by input if required. A write permit ring must be present.

The file name and enclosing parentheses may be omitted for the USE statement. In this case a scratch tape is obtained and remains scratch at the end of the program. This is the normal procedure if the tape is to be used purely as a work tape.

RECORD FORMAT

The qualifier *q* defining the format of the records on the magnetic tape must be one of the following:

FORMATTED indicating formatted records.

UNFORMATTED indicating unformatted records.

q may be omitted in which case UNFORMATTED is assumed. In special cases *q* may take the form DUMP, see *Dump and restart facilities*, page 15.

BLOCK SIZE

b is the block size, in words, of the magnetic tape blocks used to hold the FORTRAN records. In general, a large block size is more efficient but occupies correspondingly more core store. The maximum value of *b* depends on the type of tape deck but is at least 4096.

If *b* and the preceding solidus/are omitted then the block size is assumed to be 128 words.

Formatted records are held one or more to a block. Maximum record length is *b* - 2 words. Each unformatted record starts a new block, and occupies as many further blocks as are necessary to hold the whole of the record. There is no maximum record length for unformatted records. Note that four words of each block are used to hold organisational information in unformatted files.

Examples

1 OUTPUT 3 = MT1/FORMATTED (DEPT-15(1))

Channel number 3 in the program corresponds to the file DEPT-15(1) to be allocated as MT1. The block size will be 128 words.

2 USE 2 = MT3/1024

The compiler associates channel number 2 in the program with a scratch tape to be opened as MT3 for the input and/or output of unformatted records. The block size will be 1024 words.

FILE NAMES AT RUN TIME

Details of a magnetic tape or disc file to be processed may not be known at the time a program is compiled. In this case, the file name should be specified in a peripheral description statement as 'UNKNOWNASYET'. Then, the file name and other details can be provided at run time by a call of the subroutine FILE of the form:

CALL FILE (*c*, H(*n*), *g*, *r*)

where *c*, *g* and *r* are integer expressions specifying respectively the source program channel number, the file generation number and the retention period. H(*n*) is either the first of two consecutive array elements holding the 12-character file name or is a TEXT constant. The retention period is ignored and should be zero except for a magnetic tape file with a CREATE channel description statement. In the latter case, a retention period of 4095 days is recommended.

The FILE subroutine must be called before the file is opened, i.e. before any READ, WRITE or other input/output statement references the file. A call to File does not in itself open the file. Opening will take place as usual on the first other reference to that channel.

Example

A program description contains the statements

OUTPUT 7 = ED1 (UNKNOWNASYET)

INPUT 1 = CR0

The program contains the statements

READ (1, 100) FNAME (1), FNAME (2), IGEN

100 FORMAT (2A8,I3)

CALL FILE (7, FNAME (1), IGEN, 0)

The effect is to read a file name and generation number from a card; then, when there is a READ or WRITE for channel 7, the file that is opened is the one specified by this card.

FILE RENAMING

It may be convenient to open a named magnetic tape or disc file and give it a new name. In this case, the original file name must be specified in an OUTPUT, USE or CREATE statement (or possibly by a call to the subroutine FILE). Then, the new name and other details can be provided at run time by a call of the subroutine RENAME of the form:

```
CALL RENAME (c, H(n), g, r)
```

where *c*, *g* and *r* are integer expressions specifying respectively the source program channel number, the new file generation number and the new retention period. *H(n)* is either the first of two consecutive array elements holding the new 12-character file name or is a TEXT constant. For disc files the retention period is ignored and should be zero. For magnetic tape files a retention period of 4095 is recommended.

The subroutine can be called at any time. It will open the file under its original name if it is not already open, then rename it. Renaming a magnetic tape file destroys all information currently on that file.

Example

A program description contains the statements

```
OUTPUT 8 = MT2/FORMATTED (OUTFILE)
INPUT 1 = TR0
```

The program contains the statements

```
READ (1, 100) FNAME (1), FNAME (2), IGEN
100 FORMAT (2A8, I3)
CALL RENAME (8, FNAME(1), IGEN, 4095)
```

The effect is to read a file name and generation number from paper tape, open the magnetic tape file OUTFILE (if it is not already open), rename it with the new file name and generation number, and give it the standard retention period of 4095 days.

END OF FILE MARKERS IN OBJECT PROGRAM INPUT DATA

End of file markers are recognized on paper tape and cards as well as on magnetic tape and disc. In each case an execution error is flagged if an attempt is made to read beyond the end of the data.

The end of file markers for each medium are as follows:

Cards	four asterisks commencing in the first card column
Paper tape	four asterisks commencing in the first character position of the record
Magnetic tape	a trailer label (i.e. as written by ENDFILE)
Magnetic disc	an end of file bucket (i.e. as written by ENDFILE)

DUMP AND RESTART FACILITIES

Outline

The programmer can at any time call a library subroutine FDUMP which initiates a dump to either a magnetic tape or disc file. It is recommended that calls to FDUMP should be made from one point only in the FORTRAN program; a convenient point would be a user written dump subroutine which contains any other FORTRAN coding associated with dump and restart. The file dumped to must be dedicated solely to holding dumps. Full details may be found in the manual *FORTRAN: Compiler Libraries*.

Specifying the dump file

Details of the file on which the dump will be made must be specified in the program description by a statement of the form:

$$hk = xn / \text{DUMP}(f(g,r))$$

h is one of USE or CREATE.

k is the channel number (k must not be zero).

x is ED for an E.D.S. file, FD for an F.D.S. file or MT for a magnetic tape file. If h is CREATE, x must be MT.

n is the logical unit number for the file (n must not be zero in overlaid programs).

f is the dump file name and must have the form PROGRAM $\nabla abcd$ where $abcd$ are four user-defined alphanumeric characters, the first of which must be a letter.

g is the file generation number (optional).

r is the retention period; its use is optional but confined to CREATE statements.

Note: A buffer size qualifier should not be included in the statement.

More than one statement specifying a DUMP file and channel numbers may appear in the program description. On disc the file must have been allocated in the usual way (e.g. using #XJEC or #XJFC), and must have single block buckets.

PRINTING OF FORMATTED DISC AND MAGNETIC TAPE FILES

The utility programs #XRAF and #XRMF may be used to list, on the line printer, the contents of any disc or magnetic tape formatted files respectively.

There are two ways in which the programs may be used:

- 1 A listing in which a throw to head of form is made followed by each record in the file printed as a new line.
- 2 A listing in which the first character in each record is interpreted as a FORTRAN carriage control character exactly as for direct line printer output.

The specifications of #XRAF and #XRMF should be consulted for details of operating. Details of #XRAF may be found in the manual *Direct Access Utilities*, details of #XRMF may be found in the manual *Magnetic Tape Utilities*.

CHANNEL 0

Channel 0 may be used in the source program but must not be directly defined in the program description. Details are given in Chapter 7, page 33.

CONSOLE TYPEWRITER

FORTRAN programmers may use the console typewriter to output short messages to the operator with formatted WRITE statements. These messages will normally be requests, or reminders for operator action. The channel description statement will be as follows:

OUTPUT k = TYO

where k is the channel, as defined previously. The message sent to the console typewriter will vary, depending upon the particular machine, but for a multi-programming machine it will be

0 #name DISPLAY:— output record

where *name* is the four-character name of the program, and *output record* is up to 40 characters. The characters 'S] ↑ ←' must not be used in the message.

CHARACTER TRANSFERS IN CORE STORE

It is often useful to be able to reorganise the contents of a record, in character form, prior to output. For example, it may be desirable to introduce space characters into long numbers to improve the flexibility of the output. This cannot be done directly using the E,F, or I conversion codes; however, it is possible to use an array as if it were a peripheral and 'output' values to it in character form, using WRITE statements. The character handling subroutine COPY may then be used to rearrange the contents of the array before they are output to an actual peripheral using the A conversion code.

Similarly, one or more records may be read into an array using the A code in a formatted READ, then re-read using some other format.

A channel number must be associated with an array to allow the latter to be specified in READ or WRITE statements; this association is performed by a call to the subroutine DEFBUF, as follows:

CALL DEFBUF (*k*, *n*, *arrayname*)

where *k* is the channel number required (INTEGER expression), *n* is the length of the array in characters (INTEGER expression), and *arrayname* is the name of the array (of any type).

Note: An attempt to write more than one record, or to write too long a record to a DEFBUF array in a single WRITE statement, will cause execution error 121 to be flagged.

When a channel number is to be associated with a particular array at run time, the channel number must first have been defined as an array by a channel description statement of the form

USE *k* = /ARRAY

and then associated with the required array name by a call to DEFBUF. A channel number may be associated with many arrays during the running of a program, but only one at a time.

Note: Each USE statement as defined above implies one array only, regardless of how many channel numbers are represented by *k*.

Example

The statements below show how the input section of a program might be written to read a variable number of control cards of differing format. The first character determines the type of control card and is a digit in the range 1 to 9. The last control card is blank except for the digit 9 in the first character position.

```
LIST
PROGRAM (EX01)
INPUT 1 = CR0
OUTPUT 2 = LP0
USE 6 = /ARRAY
END
MASTER INPUT
DIMENSION BUFFER (10), Z(6)
CALL DEFBUF (6, 80, BUFFER)
1  READ (1, 100) BUFFER
100 FORMAT (10A8)
    READ (6, 101) K
101 FORMAT (I1)
    GO TO (51, 52, 53, 54, 55, 56, 57, 58, 59), K
50  PAUSE 77
    GO TO 1
51  READ (6, 102) X, Y, N
102 FORMAT (1X, 2F10.6, 16)
```

```

GO TO 1
52  READ (6, 103) Z
103  FORMAT (1X, 6F10.6)
GO TO 1
.....
etc
.....
59  Process data etc.

```

SUPPRESSION OF TRANSFERS

The programmer may wish to suppress transfers on some channels. This can be done by writing /NONE for Z in a channel description statement. A READ or WRITE statement referring to a channel specified in this way has no effect. Variables on an input list are not overwritten, and no action is taken on output.

Examples

```

INPUT 7 = /NONE
OUTPUT 2 = /NONE

```

READ or WRITE statements referring to channels 7 or 2 would have no effect.

This feature is useful during program testing.

If a segment being tested has some READ statements, the unit can be specified as /NONE and suitable values of variables in the read list inserted by a test segment, using a COMMON block.

Similarly, it is possible to test an output segment without producing vast quantities of useless output.

Chapter 4 Listing

COMPILER LISTING

During compilation, a listing is usually output on a line printer. The listing consists of a partial or complete list of source program statements together with other information about the program and details of any errors found. The type of listing required is specified by means of certain compiling statements known as *listing statements*.

Listing can be obtained in two degrees of detail or omitted entirely. The more detailed form is known as a *full listing*; the less detailed form is called a *short listing*.

Full listing

The first line of a full listing contains the name of the compiler, date of compilation and the time at which compilation commenced (if an internal clock is fitted). Subsequent lines list all the compiling system statements, FORTRAN source statements and compilation error messages. The length of each segment i.e. the number of object program instructions generated, is given at the end of each segment. Semi-compiled segments are listed by name only. When reading from subfiles on magnetic tape, the beginning and end of each subfile are recorded in the listing.

If the compilation is error free, a line is present listing the number of buckets used to hold the unconsolidated semi-compiled compiler output on disc.

The detection of an error during compilation causes an error message to be output on the listing. The format of these messages is given in the section *Compilation error diagnostics*, page 33.

Figure 2 on page 20 illustrates a full listing without errors.

Short listing

A short listing is similar to a full listing but does not list each FORTRAN source statement in detail; only the segment headings are listed (i.e. MASTER, SUBROUTINE, FUNCTION and BLOCK/DATA statements). In most other respects the listing is similar to a full listing. However, error messages are slightly different (see *Compilation error diagnostics*, page 33). A short list should not normally be used during the development of a program.

INITIAL LISTING STATEMENTS

The type of listing initially required is specified by an initial statement which must be one of the following listing statements:

LIST

SHORT LIST

NO LIST

If LIST is present, a full listing is output; the SHORT LIST statement causes a short listing to be output. NO LIST suppresses all listing during compilation; a line printer is therefore not required and not allocated. The NO LIST statement should be used only for fully tested programs, since if an error is found no useful information can be listed. If used, NO LIST must appear before any other statement.

A listing statement, if present, must be the first compiling system statement input to the compiler. If omitted, SHORT LIST is assumed.

INTERSEGMENT LISTING STATEMENTS

It is possible to change the type of listing between FORTRAN source segments by using LIST or SHORT LIST as an intersegment statement. In this case all subsequent listing will be of the form specified by the intersegment listing statement until another intersegment listing statement (if present) is encountered.

Intersegment listing statements are ignored if NO LIST has been specified as an initial statement. NO LIST may not be used as an intersegment statement.

```
FORTRAN COMPILATION BY #XFAE MK 3B    DATE 21/08/68

      LIST
      SEND TO (ED,ANY2THING,TE12 SC)
      PROGRAM (TE12)
      INPUT 1 = CR0
      OUTPUT 2 = LP0
      TRACE 2
      END

      MASTER TE12
      DO 1 I=1,3
      READ (1,50) A
1      WRITE (2,51) A
50     FORMAT (E6,3)
51     FORMAT (1H ,E13,6)
      STOP
      END

      END OF SEGMENT, LENGTH    36, NAME  TE12
      FINISH

      END OF COMPILATION - NO ERRORS

      S/C SUBFILE,          6 BUCKETS USED
```

Figure 2 Full listing without errors

CONSOLIDATOR LISTING

If a listing is requested from the compiler, and the compilation is error free, the consolidator will output its own listing too. The extent of the consolidator listing will depend on the mode of listing specified in the initial listing statement at the start of compilation and will be in accordance with the following table:

<i>Compilation listing mode</i>	<i>Consolidator listing mode</i>
NOLIST	*NOLIST
SHORTLIST	*SHORTLIST
LIST	*LISTNAMES

A complete consolidator listing is represented by the *LIST mode and will be generated only if the statement:

MAP

is included with the initial statements input to the compiler.

Full details of these various consolidator listing modes are to be found in the manual *Compiling Systems*.

Chapter 5 Compiler output

THE COMPILATION/CONSOLIDATION PROCESS

All successful compilations of a complete program result in the production of a compiled program in binary form. This binary program will either be left in core store ready for running or, if a DUMP ON statement (described later in the Chapter on page 23), has been specified, be dumped on a disc file for future use. The compilation/consolidation process producing the binary program is effectively a single process but involves the output of semi-compiled segments on disc as an intermediate step. These semi-compiled segments are output by the compiler and are then consolidated into the binary object program, the final output of the compilation/consolidation process.

In all successful compilations, the compiler produces semi-compiled segments on disc. A SEND TO statement (see below) is generally used to specify the subfile in the disc file to which this semi-compiled output is sent. The detection of an error during compilation causes semi-compiled output to cease but compilation continues so that further errors may be detected.

A program whose program description begins with PROGRAM or OVERLAY PROGRAM is consolidated automatically following compilation. No consolidation occurs, however, if the program description begins with SEGMENTS or OVERLAY SEGMENTS and only semi-compiled segments are output, though these segments may well be used as input to a subsequent compilation/consolidation. See *Program description statements*, page 1.

THE SEND TO STATEMENT

General

The SEND TO statement specifies the file on which the unconsolidated semi-compiled segments are to be output. In general it has one of the following forms:

SEND TO $(f(g) . s(g_s))$

SEND TO $(x, f(g) . s(g_s))$

x is either ED or FD, depending on whether an E.D.S. or F.D.S. file is to be used. If the first form of the statement is used, the file is assumed to be on E.D.S.

f is the name of a permanent file on disc which must have been previously allocated.

g is the file generation number (optional); if g and its closing parentheses are omitted, the file with the highest generation number is chosen.

s is a subfile name; if s is not specified, the compiler will choose the subfile name FORTSEMICOMP.

g_s is the subfile generation number (optional); if g_s is not specified, a generation number of zero will be given to the subfile.

The permanent file, f , which is assumed to contain no useful information, will be set up by the compiler as a new composite file.

The compiler creates a composite file consisting of two subfiles; the subfile with name s contains the semi-compiled output, while the other covers any empty space remaining in the file. If the file is found to be too short to hold all the semi-compiled output, the compiler automatically extends the file provided there is space on the disc containing the end of the file. The extension will be on this disc only and the size of the extension will depend both on the size of the file before extension and on the space available. There is a limit to the number of such extensions which can be made and this limit may vary from one version of Executive to another. Any extension which occurs during compilation will be reported on the listing device.

If there is insufficient room for extension on the disc concerned, the program will halt and the following message will be output:

0#XFAE; HALTED:-Z1

The message is not accompanied by a report and the run must be abandoned (see *Other exception conditions*, page 117).

Normal usage

This section describes the use of files for holding semi-compiled output during normal compilations (i.e. compilations where consolidation follows automatically).

USE OF THE ICLA-DEFAULT FILE

When output is to an E.D.S. file, it is not necessary to provide a SEND TO statement for normal compilations. In the absence of a SEND TO statement a work file with name ICLA-DEFAULT is assumed to be available to hold the semi-compiled output. For a discussion of ICLA-DEFAULT see *Semi-compiled output file*, page 111.

When output is to an F.D.S. file, a SEND TO statement must always be present. If the following form is used:

SEND TO (FD)

the F.D.S. file named ICLA-DEFAULT will be allocated to hold the semi-compiled output.

USE OF OTHER FILES

If for some reason, the file ICLA-DEFAULT cannot be used during a normal compilation, then the following form of the SEND TO statement which does not specify a subfile name may be used:

SEND TO ($f(g)$)

or

SEND TO ($x, f(g)$)

In this case the semi-compiled output will be held in a subfile of the file named by the user in the SEND TO statement, and the subfile will be named FORTSEMICOMP by the compiler.

SYNTAX CHECKS

To check the syntax of a program, a special form of the SEND TO statement may be used:

SEND TO (NONE)

If this statement is specified, no semi-compiled output is produced.

OUTPUT OF OBJECT PROGRAM

The compiled object program must be in binary form before it can be executed. A binary program is created by the consolidator from the unconsolidated semi-compiled segments output by the compiler. Consolidation automatically follows the successful compilation of a complete program (defined by the use of a PROGRAM or OVERLAY PROGRAM statement in the program description – see *Program description statements*, page 1).

The consolidator reads the semi-compiled segments in the disc file specified by a SEND TO statement, consolidates them into a single program and either loads the program into core store ready for running or, if a DUMP ON statement has been specified, dumps the program in binary form on a disc file. Subroutines from the FORTRAN disc compiler library file will also be incorporated at the consolidation stage. It is also possible to incorporate subroutines from other library files on disc or complete subfiles of semi-compiled produced in other compilations. (See *Semicompiled and library subfiles*, page 29).

Although the compilation/consolidation process is effectively a single procedure resulting in the production of a binary object program, two separate programs, i.e. the FORTRAN disc compiler #XFAE and the disc consolidator #XPCK, are involved. The compiler #XFAE deletes itself at the end of the compilation and loads #XPCK, passing information to the consolidator in the process. The consolidator is then automatically entered and starts consolidating. Some details of #XFAE and #XPCK may be found in Chapter 14, page 113.

THE WORK STATEMENT

The WORK statement is input as an initial statement in order to define a file to be used for work space. It has one of the following forms:

WORK ($f(g)$)

WORK ($x, f(g)$)

x is either ED or FD depending on whether an E.D.S. or F.D.S. file is to be used. If the first form of the statement is used, the file is assumed to be on E.D.S.

f is the filename.

g is the file generation number (optional).

If no WORK statement is present, a scratch file will be used.

THE DUMP ON STATEMENT

Following the compilation/consolidation process, the program may be either loaded into core store ready for turning or dumped in binary form on to a disc file. If a binary dump is required, a DUMP ON statement must be input as an initial statement. The statement takes one of the following forms:

DUMP ON

DUMP ON ($f(g)$)

DUMP ON ($x,f(g)$)

DUMP ON (x)

x can be either ED or FD depending on whether the binary dump is to be on E.D.S. or F.D.S. respectively. If x is not present, the binary dump is made to an E.D.S. file. f is the file name and should be of the form PROGRAM $\forall$ name, where name is a sequence of four letters and digits starting with a letter. If the file name is given, the binary object program is dumped into that file. If no file name is given, the binary object program is dumped into a disc file with name PROGRAM $\forall$ name, where name is the name of the object program as specified in a PROGRAM or OVERLAY PROGRAM statement.

The DUMP ON statement, if present, must appear as an initial statement preceding the program description. Not more than one DUMP ON statement should be present.

The DUMP ON statement is only relevant when a complete program is being compiled i.e. it should only be used with programs whose program description contains PROGRAM or OVERLAY PROGRAM. If the program is in fact overlaid, the DUMP ON statement is essential; otherwise consolidation of the compiled program cannot proceed and the error message

NO DUMP ON STATEMENT

will be output on the line printer. For non overlay programs the use of the DUMP ON statement is optional, dependent on whether a binary dump on disc or a program in core store is required.

THE RUN STATEMENT

In compilations where a binary dump of the object program has been made to a disc file, it is also possible to cause the object program to be loaded into core store ready for running. This is done by including the initial statement:

RUN

in addition to the DUMP ON statement.

SEPARATE CONSOLIDATION

The previous sections have emphasised that consolidation is a necessary and integrated part of the complete process of producing a binary object program. However, it should be realised that the FORTRAN disc compiler and the disc consolidator are separate programs and that the consolidator can be used as a free standing program. If automatic consolidation is not used, the consolidator must be loaded separately and presented with parameters on paper tape or cards. To avoid consolidation automatically taking place the program description should be headed SEGMENTS or OVERLAY SEGMENTS. Also LIBRARY and SEMICOMPILED statements should not be used; the information should instead be supplied directly to the consolidator.

There will be circumstances where it would be advantageous to use the consolidator #XPCK independently of the compiler. For example, the omission of a DUMP ON statement in an otherwise successful overlay program would prevent consolidation, as stated in the previous sections. Recompile could be avoided by providing the consolidator with paper tape or card parameters to effect consolidation of the already produced semi-compiled segments and generate a binary program on a disc file.

A specification of the consolidator giving full details of its use as an independent program is given in the manual *Compiling Systems*.

SOME EXAMPLES OF THE COMPILATION/CONSOLIDATION PROCESS

- 1 The following statements are to be input to the compiler:

```
LIST
SEND TO (COMFILE.SUBFAM)
DUMP ON
PROGRAM (FRED)
Program description statements
END
FORTRAN source program segments
FINISH
```

The effect produced by the compilation/consolidation process is as follows:

These statements will cause unconsolidated semi-compiled segments to be sent to a subfile SUBFAM created in an E.D.S. file called COMFILE. These segments will be consolidated into a single program called FRED which will be dumped in binary form on to a file called PROGRAM FRED.

- 2 Input to the compiler consists of the following:

```
LIST
SEND TO (NEWFILE(1). DATA(2))
PROGRAM (TEST)
Program description statements
END
FORTRAN source segments
FINISH
```

These statements will cause unconsolidated semi-compiled segments to be sent to a subfile DATA created in an E.D.S. file called NEWFILE. The segments will then be consolidated into a single program called TEST which will then be loaded into core store ready for running.

- 3 The following statements are input to the compiler:

```
LIST
SEND TO (FD, FORTSEMICOMP (10). JACK SEGS(5))
SEGMENTS (JACK)
Program description statements
END
FORTRAN source segments
FINISH
```

Unconsolidated semi-compiled segments will be output to a subfile JACK SEGS created in the F.D.S. file FORTSEMICOMP. Consolidation does not occur but these segments can be used as semi-compiled input for future compilations/consolidations.

Chapter 6 Compiler input

This chapter describes the various media from which the FORTRAN disc compiler will accept input. It also describes the different forms that input may take and describes the system in detail.

COMPILER INPUT MEDIA

The FORTRAN disc compiler will accept input from punched cards (punched in 1900 or ATLAS code), paper tape, magnetic tape or disc. The input may consist of FORTRAN source segments, semi-compiled segments or a mixture of both. Semi-compiled segments will just be copied on to the disc output file, specified in the SEND TO statement. (See *The SEND TO statement*, page 21.)

Most programs presented to the compiler are held on only one basic input medium. However it is possible to switch the input medium between segments of a program. Thus one segment may be input on cards, the next segment on magnetic tape etc. A switch in medium is specified by means of a READ FROM statement which may be present between segments of a program as an intersegment statement. The statement may also appear as an initial statement or immediately after the program description.

Initial input medium

Initially, the input medium must be paper tape or cards.

The initial input medium is indicated by the entry point to the compiler. There are three alternative entry points corresponding to the following three cases.

- 1 Start the compilation, reading initially from 8-track paper tape punched in 1900 code.
- 2 Start the compilation, reading initially from cards punched in 1900 code.
- 3 Start the compilation, reading initially from cards punched in ATLAS code.

Subsequently a switch in the input medium may be made whenever a correct READ FROM statement is encountered.

COMPILING FROM BASIC PERIPHERALS

If a complete program is not all held on the same basic medium or punched in the same code (cards only), a switch between peripherals and/or card codes may be made by a READ FROM statement. A change in basic peripheral causes the previously-used peripheral to be released by the compiler and the requested one assigned. The compiler continues reading from the new peripheral until the end of the program is reached or until another READ FROM is encountered.

In this context, the possible forms of the READ FROM statement are:

READ FROM (TR)	continue, reading from paper tape.
READ FROM (CR/ATLAS)	continue, reading from cards punched in ATLAS code.
READ FROM (CR/S1900)	continue, reading from cards punched in 1900 card code.
READ FROM (CR)	continue, reading from cards. The card code is taken to be the same as the most recent card input, established either by initial entry point or by a READ FROM (CR/ATLAS) or READ FROM (CR/S1900) statement. If all previous input was from paper tape, then 1900 card code is assumed.

Example

Consider a source program, the master segment of which, MAST, is held on paper tape. MAST calls two subroutines one of which SEG1 is on cards; the other, SEG2, was originally written for another machine and is held in ATLAS card code.

The input consists of the following

LIST	}	These statements are held on paper tape.
SEND TO (SEMFILE.SUB)		
PROGRAM (MIXI)		
INPUT 2 = TR0		
OUTPUT 1 = LP0		
END		
MASTER MAST	}	
source statements		
.....		
END	}	
READ FROM (CR)		
		When READ FROM (CR) is read, the tape reader is released and card reader assigned.
SUBROUTINE SEG1	}	These statements are held on cards in 1900 Series card code.
source statements		
.....		
END		
READ FROM (CR/ATLAS)		The same card reader remains assigned to the compiler.
SUBROUTINE SEG2	}	These statements are held on cards in ATLAS card code.
source statements		
.....		
END		
FINISH		

COMPILING FROM DISC

Source program may be input to the compiler from subfiles on disc. The subfiles concerned must form part of a standard composite file.

The READ FROM statement for disc

The programmer specifies the required subfiles to the compiler by means of a READ FROM statement of one of the following forms:

```
READ FROM (f(g))
READ FROM (f(g).s(gs))
READ FROM (x,f(g))
READ FROM (x,f(g).s(gs))
```

x is either ED or FD depending on whether the input is from an E.D.S. or F.D.S. file respectively. If x is omitted, E.D.S. is assumed.

f is the file name.

g is the file generation number (optional).

s is the subfile name.

gs is the subfile generation number (optional).

If the subfile name is omitted, the compiler will input program from the first top level subfile of the disc file.

The principal use of this form of the statement will be with source files created by the disc editor #XMED (see Chapter 10, page 51).

Compiling system statements on disc

READ FROM statements may be input from paper tape or cards but not disc. Other compiling system statements (with the exception of a NO LIST statement), may be held on disc together with the source program and input at the same time. However, it is recommended that all compiling system statements are input from a basic peripheral as this allows for greater flexibility, particularly if a compiling system statement is to be amended subsequently.

Editing of source programs on disc

Subfiles of FORTRAN source programs may be created and updated on disc by using the disc editor program #XMED. (See Chapter 10 for a full description of editing disc files.)

Example

A source program, terminated by FINISH, is contained in a subfile SOURCE on an E.D.S. file DISCFILE which has been created by XMED. It is to be compiled, consolidated and dumped to a disc file. Input consists of the E.D.S. file DISCFILE together with the following compiling system statements on cards.

```
LIST
SEND TO (OUTPUTFILE.SUB2)
DUMP ON
PROGRAM (TEST)
INPUT 2 = CR0
OUTPUT 4 = TP1
END
READ FROM (DISCFILE)
```

COMPILING FROM MAGNETIC TAPE

Both source and semi-compiled segments on magnetic tape may be input to the compiler. Magnetic tape files input to the compiler can be either composite or simple files. A FORTRAN source program or source segments must be held within a subfile in a composite file. Semi-compiled segments may also be held in a subfile, generated either by the FORTRAN magnetic tape compiler XFAM or some other magnetic tape compiler. The corresponding forms of the READ FROM statement are described in the following sections.

Input from subfiles

It is possible to compile segments held in subfiles on magnetic tape. The subfiles may contain source segments or semi-compiled segments.

The latter will merely be copied to the output file. The programmer specifies the required subfiles to the compiler by one of the following forms of the READ FROM statement:

```
READ FROM (MT, f(g) . s(gs))
READ FROM (MT, . s(gs)).
```

f is the file name.

g is the file generation number (optional).

s is the subfile name.

gs is the subfile generation number (optional).

When a file name *f* is present, as in the first type of READ FROM statement, the compiler tests to see if that file is currently open. If so, the tape is rewound and a search for the subfile *s* is made from the beginning of the file. If the named file is not open, any input magnetic tape file that is currently opened is rewound and closed. Then the file with the specified name is opened and scanned for the subfile *s*.

If the file name *f* is omitted, as in the second form of the READ FROM statement, the search for subfile *s* starts from the current position on whatever input file is already open.

The subfile *s* may contain other subfiles nested to any depth. In this case all subfiles in the nest are compiled.

On reaching the end of the subfile to be compiled, the compiler reverts back to the reader from which the READ FROM statement was input.

Subfiles containing information other than FORTRAN source or semi-compiled segments are ignored, and if a subfile of the correct name containing FORTRAN source or semi-compiled segments is not found before the end of the composite file is reached, an error message is indicated on the line printer. (See Appendix 1, page 123.) The compilation continues in error mode i.e. semi-compiled output is suppressed.

Compiling system statements on magnetic tape

READ FROM statements must not be input from magnetic tape. Other compiling system statements (with the exception of a NO LIST statement), may be held on magnetic tape together with the source program and input at the same time. However, it is recommended that all compiling system statements are input from a basic peripheral as this allows far greater flexibility particularly if a compiling system statement is to be amended subsequently.

Examples

- 1 A source program is contained on cards except for two segments held in a subfile SEGS in a composite file ANALYSIS, on magnetic tape with file generation number 1. Unconsolidated semi-compiled output is to be on a disc file OUTFILE.

Input consists of the magnetic tape file together with the following statements on cards:

```

LIST
SEND TO (OUTFILE. SUBFA)
PROGRAM (FRED)
INPUT 3 = CR0
OUTPUT 8 = MT0/FORMATTED (RESCHAP)
END
MASTER SEGA
source statements
.....
END
READ FROM (MT, ANALYSIS (1). SEGS) The compiler will start to read from magnetic tape when
                                     this statement is encountered.
FINISH

```

The compiler will read from cards until the READ FROM statement is encountered. Then it will compile from magnetic tape until the end of the subfile SEGS is reached. At this stage it will revert to reading from cards, i.e. the FINISH statement.

- 2 Each segment of a program is contained in a separate subfile on magnetic tape. The names of the subfiles in the order in which they appear on the composite file IPFILE are PROGDES (containing compiling system statements), SEG1, SEG2, SEG3. The compiled segments are to be output to a disc file FILEOUT. The input consists of the magnetic tape file together with the following statements on punched cards:

```

LIST
SEND TO (FILEOUT . SUBFILE ONE)
READ FROM (MT,IPFILE.PROGDES)
READ FROM (MT,..SEG1)

```

```

READ FROM (MT,.SEG2)
READ FROM (MT,.SEG3)
FINISH

```

Input from simple files

A complete simple file may be input by a statement of the form

```

READ FROM (MT, $f(g)$ )

```

where f is a simple file name and g the file generation number. Only a consolidated semi-compiled program may be input from a simple file. This type of READ FROM statement will probably be used for the FORTRAN disc compiler only to incorporate semi-compiled segments produced by a non-FORTRAN magnetic tape compiler.

Editing of source programs on magnetic tape

Source programs may be created and updated on magnetic tape by means of one of the editing programs #XKYA or #XMUM. (See the manual *FORTRAN: Magnetic Tape Compiler*.)

SEMI-COMPILED AND LIBRARY SUBFILES

During the consolidation process, when the semi-compiled segments produced from the FORTRAN source program together with any necessary routines from the FORTRAN compiler library are consolidated in order to generate an object program, it may be desirable to input other semi-compiled segments from disc files.

These semi-compiled segments will be either output from previous compilations of source segments written in either FORTRAN or in other programming languages (see Chapter 11, page 57) or segments contained within standard libraries other than the FORTRAN compiler library.

The files and subfiles containing any additional semi-compiled segment to be input are specified to the compiler by means of a LIBRARY or SEMICOMPILED statement. The two statements correspond to two different modes of input during consolidation.

The LIBRARY statement is used when only those segments specifically required in the program are to be incorporated from the file or subfile by the consolidator, i.e. the consolidator is accepting segments selectively. The scanning of the FORTRAN library block is an example of selective input to the consolidator.

The SEMICOMPILED statement is used when all the semi-compiled segments contained within the specified subfile are to be consolidated into the binary object program regardless of whether they are required or not, i.e. the consolidator is accepting segments inclusively. The consolidation of the semi-compiled segments in the FORTRAN disc compiler output subfile is an example of inclusive input to the consolidator.

The SEMICOMPILED statement

The SEMICOMPILED statement is an initial statement and, if present, causes the consolidator to incorporate into the object program all the semi-compiled routines contained within the specified disc subfile. These routines may consist of previously compiled segments written by the user in either FORTRAN or other programming languages. Mixed language programming is described in Chapter 11.

The statement has one of the following forms:

```

SEMICOMPILED ( $x, f(g).s(g_s)$ )

```

```

SEMICOMPILED ( $f(g).s(g_s)$ )

```

x is either ED or FD, depending on whether an E.D.S. or F.D.S. file is to be used. If the second form of the statement is used, the file is assumed to be on E.D.S.

f is the file name.

g is the file generation number (optional).

s is the subfile name.

g_s is the subfile generation number (optional).

The semi-compiled routines contained in the subfile s of the file f are all to be incorporated in the object program, i.e. any such subfile is treated by the consolidator in exactly the same way as the semi-compiled output subfile produced as the result of a SEND TO statement.

The effect of a SEMICOMPILED statement is similar to that of the equivalent READ FROM statement when reading semi-compiled. However, in the case of the READ FROM statement, the semi-compiled segments would be copied to the compiler output file.

THE LIBRARY STATEMENT

The LIBRARY statement is an initial statement and if present, causes the consolidator to scan a semi-compiled library on disc, incorporating only those semi-compiled library routines required by the program.

The statement has one of the following forms:

LIBRARY ($f(g)$)

LIBRARY ($f(g).s(g_s)$)

LIBRARY ($x,f(g).s(g_s)$)

LIBRARY ($x,f(g)$)

x is either ED or FD depending on whether the library is on E.D.S. or F.D.S. respectively. If x is not present, the library is assumed to be on E.D.S.

f is the file name.

g is the file generation number (optional).

s is the subfile name.

g_s is the subfile generation number (optional).

In the forms of the statement where the subfile name is not specified, the first subfile in the file f will be scanned.

A LIBRARY statement is not necessary in order to scan the FORTRAN disc compiler library. This search, incorporating the various standard FORTRAN functions and subroutines, is carried out automatically.

Use of the LIBRARY and SEMICOMPILED statements

Both the LIBRARY and SEMICOMPILED statements, if present, must be input as initial statements; otherwise they will be ignored. There is no restriction on the number of LIBRARY and SEMICOMPILED statements that may be present in any one program.

With the exception of the semi-compiled subfile produced during compilation, the order of searching and incorporation of specified files or subfiles will be in the order of occurrence of the various compiling system statements. The semi-compiled segments in the subfile specified by the SEND TO statement will be incorporated in the object program first, regardless of the position of the SEND TO statement, followed by other files and subfiles in the order in which the corresponding LIBRARY and SEMICOMPILED statement (if present) occur. The FORTRAN compiler library will be searched last. This library must be held in a disc file with the name SUBGROUPSRF7. However, if it is desired to search this library earlier, a LIBRARY statement of the form LIBRARY (SUBGROUPSRF7) should be input in the appropriate position.

Example

The following set of compiling system statements are input to the compiler on cards.

```
LIST
SEND TO (FORTSEMICOMP. EXAM SC)
DUMP ON (PROGRAM FILE)
SEMICOMPILED (ANY FILE. EXAMSEGMENTS)
LIBRARY (SUBGROUPSRA1)
PROGRAM (EXAM)
Program description statements
.....
END
MASTER
```

FORTTRAN source statements

END

etc

During compilation, the semi-compiled segments generated from the source segments will be output in a subfile EXAM SC in the file FORTSEMICOMP. The compiler will then pass on information to the consolidator causing it to carry out the following sequences of events.

- 1 Consolidate the segments in subfile EXAM SC
- 2 Consolidate the segments in the E.D.S. file ANY FILE, subfile EXAMSEGMENTS. This subfile of semi-compiled segments may have been created by a previous compilation.
- 3 Scan the library routines contained in the file SUBGROUPSRA1 on E.D.S. incorporating those routines required.
- 4 Scan the FORTTRAN library contained in the file SUBGROUPSRF7, incorporating those routines required.
- 5 Produce a binary version of the object program in the file PROGRAM FILE.

When it is necessary to search library groups other than the SRF7 group, it is advisable to do so in the following order:

- 1 Search private libraries containing segments which have been written by the programmer or installation.
- 2 Search the FORTTRAN library SRF7.
- 3 Search any other library containing ICL-written segments; for example, the S-RS library.
- 4 Search the FORTTRAN library again (the compiler takes care of this last search automatically).

If step 3 is omitted, then step 2 is not required. If step 1 is omitted, step 2 is still advisable; otherwise wrong versions of certain routines may be included (for example, if SRA1 is scanned before the SRF7 group, the Algol version of SIN will be included).

Example

If a source program contains calls to private routines in a library, which in turn call routines in the S-RS groups, the initial statements would be as follows:

LIBRARY (MYLIBRARIES.PRIVATE LIB)

LIBRARY (SUBGROUPSRF7)

LIBRARY (SUBGROUPS-RS)

Notes:

- 1 A final LIBRARY (SUBGROUPSRF7) is not needed in the above example as the compiler supplies it automatically.
- 2 The libraries in the example are assumed to be on E.D.S. For a library on F.D.S., FD must be specified; for example,

LIBRARY (FD,SUBGROUP-RS)

Chapter 7 Program testing

COMPILATION ERROR DIAGNOSTICS

During compilation of a FORTRAN program, a partial or complete list of source program statements is output on the listing peripheral. When an error is found, an error message is output. If a full listing has been specified, the error message follows the complete statement. In a short listing, the error message is preceded by the first line only of the statement in error; the error could be in a continuation line. The detection of an error causes semi-compiled output to cease; the compiler continues so that further errors may be detected.

A list of compilation error numbers is given in Appendix 1, page 123.

Error messages

The format of the error message is normally:

ERROR n IN LAST STATEMENT, LINE p , CHAR q

n is the error number.

p is the line number within the statement (starting at line 0).

q is the character number within the line (starting at character 1).

The error is most likely to have occurred at or before the position specified by p and q .

The occurrence of one error may lead the compiler to produce further spurious errors later in the program.

Consider the following sequence:

```
DIMENSION A(10), Z(5,70), AMISH(7,10*,LOOK(5)
```

```
.....
```

```
READ(UNIT,5) A, I, LOOK
```

```
.....
```

```
WRITE(UNITA,6) A(1), I, LOOK(3)
```

The character) has been mispunched as *. The DIMENSION statement will be faulted because of the *; the compiler will not register LOOK as an array but will treat it as a variable in the READ statement: then in the WRITE statement an error will be noted because a subscript is attached to a variable. The reasons for such spurious errors are not always easy to find. If the cause of certain error lines cannot be ascertained, then obvious errors should be corrected and the program recompiled; any spurious errors will then disappear.

LABEL ERRORS

Errors 15 and 16 occur when a label referred to in a segment is not present. These errors will not be found until the end of the segment is reached. Messages of this type have a different format:

ERROR n - LABEL m

where m is the label not found. Messages referring to label 0 should be ignored as they indicate only that an error has already been found in some statement that refers to a label.

MISSING SEGMENTS

If some segments of a program are not input to the compiler, a loadable program is still produced, but the message SEGMENTS MISSING will be output on the listing. An incomplete program should be run only if it is known that the missing segments will not be entered on that run. Any attempt to actually enter a missing segment will result in execution error 104 being reported. The program cannot be continued even if an error trap routine is present, see *The error trap facility*, page 35.

OBJECT PROGRAM ERROR DIAGNOSTICS

The discovery of the causes of run-time error conditions can be an expensive process, involving wastage of machine time and programming effort. For this reason it has become standard practice to incorporate monitoring routines into programs; these routines carry out tests during run-time and output details of any error conditions that occur. All compiled FORTRAN programs contain monitoring routines and are said to be *traced*; the necessary routines are incorporated by the compiler and no extra programming is required.

The advantages of tracing are obtained at the cost of increased run-time and core store requirements. The amount by which these two factors are increased is determined by the level of tracing used; i.e. by the comprehensiveness of the tests carried out by the monitoring routines. The higher the level of tracing the longer will be the run-time and the greater will be the core store occupied.

The level of tracing that it is desirable to incorporate into a program is determined by the relative importance of the two conflicting requirements – run-time efficiency and ease of error finding. During the early stages of the development of a program ease of error finding will be the important requirement and a high trace level will be desirable. When the program has been fully tested and proved, a lower trace level may be used with a corresponding increase in run-time efficiency. To provide for this variation three levels of tracing are available, 0, 1 and 2, being low, medium and high levels respectively. It is envisaged that trace level 2 will be used during program development and trace level 1 during normal production runs. Trace level 0 will be used only for well-established programs and when core store and execution time are at a premium.

Trace output channel

Trace information will normally be output on the first output channel specified in the program description. The user should ensure that this channel will accept formatted output, as trace information must not be output to an unformatted magnetic tape file, for instance. The trace output channel may be used for other, non-trace information, as the user chooses. The user may output non-trace information on the trace output unit, without knowing its channel number, by using the channel number 0 in WRITE statements.

Example

```
PROGRAM (JIM5)
INPUT 9 = TR0
OUTPUT 3 = LP0
OUTPUT 5 = TP1
END
```

The trace information would be output on the line printer, also used as the programmer's unit 3.

Trace level 1

Level 1 is the normal mode of tracing and is automatically used unless the programmer includes a TRACE statement in the program description (see below), specifying a different level. The effects of tracing at level 1 are described below.

NORMAL ERROR ACTION

The occurrence of a detectable error causes the following message to be output on the trace output device:

EXECUTION ERROR *n* PROGRAM TERMINATED

where *n* is an error number. A full list of error numbers is given in Appendix 2, page 127. This line is followed by a list of the last 25 segment entries and exits under the heading SEGMENT TRACE. An entry to a segment is marked by the first eight characters of the segment name and an exit is marked by RETURN. In this context, the term 'segment' means a segment compiled from FORTRAN source, a Basic External Function, or any PLAN segments incorporated in a FORTRAN program using FPROLOG and FEPILOG. (See Chapter 11, page 57). The program then halts, displaying the characters EE on the console typewriter.

OVERFLOW

Overflow may occur if the result of an arithmetic expression or the result of evaluating part of an arithmetic expression is out of range, or if the program attempts to carry out some mathematically undefined action such as division by zero. Overflow may also occur during the evaluation of a logical expression that compares two arithmetic expressions if the difference between the two is out of range; this is most likely to occur during the evaluation of logical IF statements.

If overflow occurs, an overflow indicator is set and the program continues. A test on the state of this overflow indicator is made prior to an entry to or an exit from every compiled subroutine or function segment and every Basic External Function. If the overflow indicator is set at such a point, execution error number 50 is output.

User's check

A subroutine is provided in the compiler's library that enables the user to check the state of overflow in his program. The subroutine, called OVERFL, requires one integer argument and is called in the standard way:

```
CALL OVERFL (j)
```

j may be any integer variable or array element. The actual argument is assigned the value 1 if overflow is currently set, or the value 2 otherwise. The subroutine leaves overflow clear. This subroutine is provided chiefly to allow programs written for another system to be run on a 1900 machine. It may also be used to test for overflow more frequently than the automatic facility of trace level 1.

THE ERROR TRAP FACILITY

The user may augment the standard trace facilities by providing a subroutine to be entered whenever an error is discovered by the trace routines. Such a subroutine is called an *error trap*. An error trap may be written to output specialised information chosen by the user rather than by the system. Alternatively, in certain cases the subroutine may be written to carry out some remedial action enabling the program to continue.

An error trap is set up by calling the library subroutine FTRAP with the user's error subroutine as an actual argument. Thus, if the user's subroutine is called PMORTM, the user would write:

```
EXTERNAL PMORTM
```

```
.....
```

```
.....
```

```
CALL FTRAP (PMORTM)
```

The subroutine FTRAP need be called once only in a program and will normally be called at the beginning of the program.

The subroutine PMORTM must be written with one dummy argument of type INTEGER. When an error is detected, the subroutine will be entered with this argument set equal to the error number as listed in Appendix 2. (There is one special case dealt with below.) Thus, the subroutine PMORTM might be written:

```
SUBROUTINE PMORTM (ERR NO)
```

```
INTEGER ERR NO
```

```
COMMON/ABCD/ARRAY (10),X,Y,Z
```

```
WRITE (0,101) ERR NO, ARRAY,X,Y,Z
```

```
101 FORMAT (1H∇,6HERROR∇,12/1H∇,10E10.4/1H∇,3E10.4)
```

```
STOP
```

```
END
```

This subroutine outputs the error number, followed by the current values of an array and three variables, on the trace output channel (i.e. channel 0).

If the error consists of an unacceptable character in a numeric input field (i.e. corresponding to error 0 'x' as listed in Appendix 2) then the argument of the error subroutine is not set to an error number, but is set to the negated internal representation of the character; i.e. if the argument is negative, it should be made positive and its text value will then be 4H000x, where *x* is the erroneous character. The above example could be expanded to take care of this as follows:

```
SUBROUTINE PMORTM(ERR NO)
```

```
INTEGER ERR NO
```

```
COMMON/ABCD/ARRAY (10),X,Y,Z
```

```
IF (ERR NO.GT.0) GO TO 5
```

```
ERR NO = - ERR NO
```

```

WRITE (0,100) ERR NO,ARRY,X,Y,Z
STOP
5 WRITE (0,101) ERR NO,ARRY,X,Y,Z
100 FORMAT (1HV, 7HERROR ' V,A4, 1H' /1HV, 10E10.4/1HV,3E10.4)
101 FORMAT (1HV,6HERRORV,I2/1HV,10E10.4/1HV,3E10.4)
STOP
END

```

If the programmer requires the program to continue after an entry to the error trap subroutine, the subroutine can end with a RETURN statement rather than a STOP statement. For non-fatal errors, control will then return to the point in the program where the error was detected and the operation that caused the error will have a meaningless result. For fatal errors it is not possible for the program to continue and a RETURN statement will cause the program to output the standard postmortem information and halt. The distinction between fatal and non-fatal errors is given in Appendix 2. The user may set up a different error trap at any point in the program by calling FTRAP with a different argument. Alternatively the standard error procedure may be reinstated by calling the subroutine FRESET as follows:

```
CALL FRESET
```

This subroutine cancels any previous call to FTRAP.

Trace level 2

Trace level 2 is the highest level of tracing available. All trace level 1 facilities are also provided in trace level 2; they will not be described again in this section.

Trace level 2 is incorporated by including in the program description the statement:

```
TRACE 2
```

ERROR ACTION

The postmortem information output on the detection of an error is the same as for trace level 1 with the addition of a STATEMENT TRACE, which gives details of the last 100 statements executed preceding the error.

STATEMENT TRACE output

The STATEMENT TRACE list is output following the standard error line and contains a line for each of the 100 statements examined. Each line contains the following details in the order shown:

The line number, in the range - 100 to - 1 inclusive.

The statement label, if one is present.

An abbreviation of the statement type.

The result of the statement, if applicable.

The end of a DO loop, although not written as a statement in the source program, is treated as a statement (type LOOP).

The statements, their abbreviations and their results are shown in Table 2, page 37.

Note: Whenever control has been passed to a new segment, the name of that segment is printed on a separate line. In the case of an overlaid object program (see *The overlay system*, page 43), these names will not be correct if the segment entered has subsequently been overlaid by a segment in another unit of the same overlay area.

<i>Statement</i>	<i>Abbreviation</i>	<i>Result</i>
Arithmetic assignment	ARTH	Value of left-hand side
Logical assignment	LOGC	Value of left-hand side: TRUE or FALSE
Arithmetic IF	IF	Value of expression in parentheses
Logical IF	LIF	Value of logical expression: TRUE or FALSE
GO TO	GO	
DO	{ DO LOOP	Initial parameter (terminal parameter) – (control variable)
Computed GO TO	CGO	Control variable
Assigned GO TO	AGO	
READ	READ	
WRITE	WRTE	
PAUSE	PAUS	
STOP	STOP	
CALL	CALL	
RETURN	RETN	
ENDFILE	ENDF	
CONTINUE	CONT	
BACKSPACE	BACK	
REWIND	REW	
ASSIGN	ASGN	

Table 2 Statements in a STATEMENT TRACE list

ARRAY ELEMENTS

The start and end addresses of each array are held in store at object program run time. At trace level 2 a check is made that the generated address of an array element falls within the bounds of the array. No check is made on individual subscripts. For example, if ARRAY (3,2) is declared, then a reference to ARRAY (5,5) would be faulted. However, ARRAY (4,1) would not be detected as an error. The corresponding error number is 01. (See Appendix 2, page 127).

TRACE 2 WITH STEERING LISTS

It may be desirable to obtain detailed trace information about selected parts of a program. If the program was compiled at trace level 2 it is possible to selectively output information about parts of the program by specifying these parts in a *trace steering list*. The trace steering list is read from the trace input channel, usually a card or paper tape reader, at the start of the program run. The trace input channel is the first input channel specified in the program description. The user must ensure that it is capable of inputting formatted data.

Example

```

PROGRAM (FRED)
INPUT 5 = TR0
INPUT 7 = CR0
OUTPUT 6 = LP0
TRACE 2
END

```

This program description will cause the steering list to be read, at run time, from a tape reader allotted as TR0. The steering list would be input from this unit before any data associated with the programmer's channel 5.

Format of trace steering lists

A trace steering list consists of one or more lines, each of which specifies a continuous sequence of statements, within a segment, that is to be traced. Each sequence of statements is defined by specifying the beginning of the sequence relative to a *trace triggering statement*, which must be labelled, and the length of the sequence.

Each line of a trace steering list has the following format:

$l \quad d \quad c$

l is the label of a statement that is to be the trace triggering statement for the traced statements. The same label must not be listed more than once. d , where $-99 \leq d \leq 2047$, indicates at which statement, relative to the triggering statement, tracing is to commence. For example, if d is 50, trace information is output when 50 statements following the triggering statement have been executed.

It should be noted that d does not refer to the number of statements *written* after the label, but to the number that have actually been executed. If d is 0, trace output starts with the triggering statement itself. Negative values of d indicate that statements executed before the triggering statement are to be traced; however, no trace information is actually output until the triggering statement itself is executed.

c , where $0 \leq c \leq 4095$, is the total number of statements to be traced for that triggering statement.

If there is an overlap between the ranges specified by two steering lines, fewer than c lines may be output, since the output from the first statement will cease when the second triggering statement is encountered. If the value of d for the second triggering statement is negative, and overlaps the scope of a previous triggering statement, some information may be repeated.

The format of a trace steering list on a FORTRAN coding sheet is as follows:

<i>segment name</i>	}	
$l_1 \quad d_1 \quad c_1$		
$l_2 \quad d_2 \quad c_2$		
$l_3 \quad d_3 \quad c_3$		
.....		
$l_n \quad d_n \quad c_n$		
END		
FINISH		

repeated for each segment

The segment name, END and FINISH must begin in or after column 7.

Columns 73 to 80 are ignored and column 6 should be blank.

l should be written in columns 1 to 5.

d should begin in column 7.

c should begin in column 13 or be preceded by a tab character.

No spaces are allowed in c and d . Tabs are allowed on paper tape in the same way as in source program.

A STATEMENT TRACE list is output for each series of statements specified in the trace steering list. Line numbers are consecutive in ascending order, in the normal manner. The trigger statement line is always numbered 0; line numbers for statements executed before the trigger statement are given negative numbers.

TRACING EVERY STATEMENT

The user can obtain trace information for every statement that has been compiled with trace level 2, without specifying a trace steering list, by activating the program at entry point 27. (See *Loading and running the object program*, page 115.)

Suppression of trace output

Trace output during the running of a program may be suppressed by switching off switch 0, either by a call to the subroutine SWOFF or by operator action. The normal error information will still be output if an error occurs while switch 0 is switched off.

Trace level 0

If core store is at a premium, a zero level of tracing is available that saves about 200 words of core store compared with TRACE 1. The limited facilities provided form a subset of the trace level 1 facilities. On the detection of an error, the standard error line is output giving the error number. However, no SEGMENT TRACE list follows. There is no user-error trap facility and no test on the state of the overflow register. The detection of any error always causes the program run to be terminated. Trace level 0 may be requested by writing, in the program description, the statement,

TRACE 0

Mixed error detection levels

During development of a program, it may be convenient to run part of it at level 1 and part at level 2. This can be done by specifying TRACE 2 in the program description and then TRACE 2 or TRACE 1 between program segments as required. Each segment is compiled at the level set by the previous TRACE statement. A trace steering list must not refer to a segment that has been compiled at level 1. Statement lines output under the heading STATEMENT TRACE will include only those statements traced at level 2.

Operator intervention

The operator may intervene at program run time and cause tracing information to be output. The normal reasons for this are:

- 1 The program has exceeded its time limit.
- 2 Executive has detected an illegal machine code instruction causing an ILLEGAL message to be printed on the console typewriter.

Trace information output takes the same form as if an error had been detected by a trace routine in the program except that no executive error line appears. Details of the necessary console messages are given in the section *Loading and running the object program*, page 115.

User defined errors

It may be required to obtain the standard post mortem or error action as a result of an error detected in the user's program. The subroutine FERROR is available in the library for this purpose.

The statement

CALL FERROR(*n*)

where *n* is an integer variable or constant in the range 900 to 999 may be included anywhere in a program and it will cause the standard error action (depending on the trace level) to take place; execution error *n* will be reported. If a user error trap routine is present then control will pass to this subroutine as for a standard execution error.

If *n* takes the special value 999, no execution error will be reported; instead, the standard post mortem will be given, all peripherals will be released and the program will halt EE. This facility would be useful in an error trap routine.

If *n* is outside the permitted range, execution error 47 will be reported.

Example

If a variable POS should never be negative at some point in the program then the statement

IF(POS.LT.0.0)CALL FERROR(918)

would initiate error action if POS were negative.

Example of error detection in a program with an arithmetic error

The following program EREX reads a list of numbers J_i , A_i , B_i ($i = 1, 2, \dots, n$) into the blank COMMON block.

A check is made to set $A_i = -1$ if $\sqrt{(A_i - B_i)} \leq 3.142$. Subroutine PROCES is then called to operate on these numbers (the action of the subroutine is immaterial to the example).

```

LIST
PROGRAM (EREX)
OUTPUT 2 = LP0
INPUT 1 = TR0
TRACE 2
END
MASTER READ
COMMON J(10), A(10), B(10), N
2 READ (1,100) N
IF (N.EQ.0) STOP
WRITE (2,200) N
DO 1 I = 1, N
READ (1,101) J(I), B(I)
IF (SQRT(A(I)-B(I)).LE.3.142) A(I) = - 1.0
1 CONTINUE
CALL PROCES
GO TO 2
100 FORMAT (12)
101 FORMAT (12, 2F6.1)
200 FORMAT (14H1PROGRAM #EREX//13HVNOV OFV VALUES, I4//)
END
SUBROUTINE PROCES
.....
END
FINISH

```

The following output on the line printer shows what may happen if no error trap is used and there is an error in the input data. Error 57 is reported (negative argument in SQRT), STATEMENT TRACE shows how far the program has gone, and SEGMENT TRACE shows that SQRT was the last routine called.

PROGRAM#EREX			
NO OF VALUES 8			
EXECUTION ERROR 57-PROGRAM TERMINATED			
STATEMENT TRACE			
READ			
-22	2	READ	
-21		LIF	FALS
-20		WRTE	
-19		DO	1
-18		READ	
-17		LIF	FALS
-16	1	CONT	
-15		LOOP	6

-14	READ		
-13	LIF	FALS	
-12	1 CONT		
-11	LOOP		5
-10	READ		
- 9	LIF	TRUE	
- 8	ARTH	-1.00000000000E	0
- 7	1 CONT		
- 6	LOOP		4
- 5	READ		
- 4	LIF	FALS	
- 3	1 CONT		
- 2	LOOP		3
- 1	READ		

SEGMENT TRACE

SQRT
 RETURN
 SQRT
 RETURN
 SQRT
 RETURN
 SQRT
 RETURN
 SQRT

Example of error trap facility

An error trap subroutine is incorporated in the program. The program description is as in the previous example. The MASTER segment begins:

```

MASTER READ
COMMON J(10), A(10), B(10), N
EXTERNAL ERROR
CALL FTRAP (ERROR)

```

2 READ etc. as in the previous example

The error subroutine is

```

SUBROUTINE ERROR(K)
COMMON J(10), A(10), B(10), N
WRITE (2,100) K, (J(I), A(I), B(I), I = 1,N)
RETURN
C RETURN IN ORDER TO CHECK REST OF DATA
100 FORMAT (/6HERROR, I3, 9HDETECTED//22HVALUES IN COMMON AREA
1 /// (3X, I2, 2F6.1/))
END

```

The output consists of the following on the line printer:

PROGRAM#EREX		} output by the program
NO OF VALUES 8		
ERROR 57 DETECTED		
VALUES IN COMMON AREA		output by subroutine ERROR
74	153.1	78.6
13	102.8	84.7
25	-1.0	97.1
31	177.9	53.8
65	94.5	98.4
0	0.0	0.0
0	0.0	0.0
0	0.0	0.0

The program continues with the rest of the data.

Chapter 8 Overlays

THE OVERLAY SYSTEM

Programs may be written which are too large to be held in the core store of the computer. Such a program is run by *overlaying* parts of the program, from a backing store medium. That is, only part of the program is in the core store at a particular time, the remainder being held on the backing store and copied into the core store whenever it is required.

To use the overlay system, the programmer must divide his program into units, one unit which is not to be overlaid and a number of *overlay units*. The section of program which is not to be overlaid is called the permanent unit and is kept in the *permanent area* of core store. It contains the MASTER segment and any other segments that the programmer chooses. The permanent unit will also contain various subroutines that are inserted into a FORTRAN program by the compiler and not referred to explicitly by the programmer (e.g. the input and output subroutines). Each overlay unit of the program must be assigned to an *overlay area* of core store. There may be several overlay areas and each one may have several overlay units assigned to it. At any one time an overlay area may contain only one of the units assigned to it.

The structure of an overlaid program is defined in OVERLAY statements in the program description. This type of statement is described below.

1900 FORTRAN does not require the user of the overlay system to call any special subroutines. The source program is written in the normal way and segments are called by CALL statements or function references exactly as if the whole program was to be in the core store.

If the segment called is in an overlay unit, the system checks to see whether that unit is already in the core store. If it is not, it will be copied in from the backing store, then entered normally. A RETURN statement in the segment will have a similar effect, checking to see whether the segment containing the calling statement is in the core store, and copying it in if it is not. Decisions on overlay organisations are not, therefore, required at the time of writing a source program. This organisation need not be defined until the program is about to be compiled.

PROGRAM DESCRIPTION STATEMENTS FOR OVERLAY PROGRAMS

The OVERLAY statement

The form of this statement is as follows:

OVERLAY (*a,u*)*s*₁, *s*₂, *s*₃, ..., *s*_{*n*}

a is the overlay area number in the range $1 \leq a \leq 255$.

u is the overlay unit number in the range $1 \leq u \leq 1023$.

*s*₁, *s*₂, *s*₃ ..., *s*_{*n*} is a list of the names of segments to be included in unit *u* of area *a*.

The numbers used for *a* and *u* need not be consecutive. The same unit numbers can be used for units in different areas. Since the OVERLAY statement is a program description statement, no continuation lines are possible. However, if the same area and unit numbers appear in more than one statement, then the unit will comprise all the segments named in all such statements. For example, the statements:

OVERLAY (3,7) SUBA1, SUBA2, FUNC7

OVERLAY (3,7) SUBB3, SUBB5

will have the same effect as:

OVERLAY (3,7) SUBA1, SUBA2, FUNC7, SUBB3 SUBB5

All segments that have not been assigned to an overlay unit will be assigned to the permanent unit. The permanent unit is not specified explicitly by the programmer.

The OVERLAY PROGRAM and OVERLAY SEGMENTS statements

The OVERLAY PROGRAM statement must be used in place of a PROGRAM statement in order to introduce an overlay program. It has the following form

OVERLAY PROGRAM (*name*)

where *name* is the program name.

The OVERLAY SEGMENTS statement is used instead of a SEGMENTS statement in order to introduce segments to be compiled which may later be included in an overlay program. It has the following form

OVERLAY SEGMENTS (*name*)

where *name* is as above. The area and unit number of each overlay segment is not required until the complete program is compiled and therefore OVERLAY statements should not appear in a program description introduced by OVERLAY SEGMENTS.

OVERLAY PROGRAM may be used to introduce a program with no overlays if it is wished to include segments which were compiled under an OVERLAY SEGMENTS statement, but a slight loss in core store usage is involved in doing this. This situation of overlay segments in a non-overlay program would arise if a library of segments is required for use in either overlay or non-overlay programs.

The DEPTH OF OVERLAY statement

At run time when a segment A of an overlay program in one unit calls a segment B in another unit, the following three actions take place:

- 1 The new overlay unit is brought into core store (if not already there).
- 2 Control is transferred to segment B.
- 3 A record is kept of the unit and area numbers of segment A so that when a return is made to segment A the correct unit can be brought in.

This situation becomes more complex if B calls a further segment C in another unit (possibly even the unit that segment A is in). In this case records of the unit and area numbers of A and B must be kept. This process can go much further in some programs and a list of these records must always be kept.

The space reserved by the compiler to contain this list of records must be sufficient to contain the maximum number of segment entries required.

The size of the list is specified by means of a DEPTH OF OVERLAY statement. The *depth of overlay* is defined to be the maximum number of calls (or function references) which can be obeyed *in sequence*, and which each transfer control to a new overlay unit. A record is kept in the list for each of these calls. Here 'unit' includes the permanent unit.

The statement

DEPTH OF OVERLAY *n*

in the program description, where *n* is an integer, implies that the list will contain a maximum of *n* records, i.e. the statement will reserve storage for *n* entries in the list. In the absence of the DEPTH OF OVERLAY statement, *n* is set equal to the total number of units in the program. This is usually an over estimate so that the statement is obligatory only in the case of highly segmented programs. On the other hand, it is useful to save core store by specifying an appropriate length for the list when this is very much less than the number of units.

Examples

- 1 A program contains 40 overlay units and one overlay area. All calls to overlay units are made from the permanent unit and an overlay unit does not call another overlay unit, but may call standard functions in the permanent unit. The programmer may save the space of 38 entries in the list of records by including the statement:

DEPTH OF OVERLAY 2

in the program description.

- 2 A program contains two overlay units and two overlay areas. A segment in the permanent unit calls a segment 1 of unit A, which calls segment 1 of unit B, which in turn calls segment 2 of unit A etc. There are 20 calls in all between unit A and unit B. Segments in units A and B call standard functions which are held in the permanent unit. In order to reserve enough storage to hold the list of records the programmer *must* include the statement:

DEPTH OF OVERLAY 22

in the program description.

Restrictions

The following restrictions must be observed in overlay programs:

- 1 A CALL or function reference must not bring down another unit to overwrite the unit containing the CALL or the function reference, whether this is done directly or indirectly via a chain of CALLs and function references. There is one exception: a CALL with no arguments may bring down a unit to overwrite itself; in this case, however, a RETURN may not be made unless the calling segment also has no arguments.
- 2 No subroutine or function whose name is listed in an EXTERNAL statement may be in an overlay unit.
- 3 No segment name may appear in two OVERLAY statements.
- 4 EDO (or FDO) must not be used by an overlay program.

It should be noted that non-common variables, array elements, and arrays will be overlaid if assigned values in DATA statements. These variables, array elements and arrays will be reset to the specified values each time the overlay unit is brought in.

COMPILATION

The structure of an overlay program must be defined in the program description each time the program is compiled, whether or not the structure has changed. If the structure is to be different from the previous compilation it is possible to re-compile by providing a new complete program description and re-inputting the semi-compiled program produced by the previous run.

OBJECT PROGRAM OPTIONALLY OVERLAID FROM MAGNETIC TAPE

Overlay programs compiled by #XFAE are normally overlaid from disc during execution of the object program. If it is required to overlay the program from magnetic tape, the following statement must be included in the source program (a suitable place is immediately after the MASTER statement):

EXTERNAL OLAYMT

The object program will be produced on a disc file as specified in the DUMP ON statement. See *The DUMP ON statement*, page 23. Before the program can be run it must be transcribed to magnetic tape using #XPEU and #XPEC; details of these routines may be found in the manual *Compiling Systems*.

RUN TIME EFFICIENCY

The most efficient utilisation of available core store will be obtained if each overlay unit associated with a particular overlay area is approximately the same length. It is not possible to estimate exactly the number of machine code instructions that will be generated from a FORTRAN source segment, but during compilation the compiler outputs to the listing peripheral a line at the end of each segment, which specifies the program area used by the segment. To this must be added the space taken up by constants in DATA statements. (Note that if a single element of an array occurs in a DATA statement, the whole space of the array is included in the size of the overlay unit.)

Lastly, the structure of overlays should be designed so that units seldom need be interchanged.

STANDARD FUNCTIONS IN OVERLAYS

Standard functions and subroutines will normally be in the permanent area. The programmer can, if he wishes, assign a standard function or subroutine to an overlay unit, but there is not much to be gained by doing so since these routines are small segments making use of system segments which are always in the permanent area.

EXAMPLE OF AN OVERLAY PROGRAM

A program with the following structure is to be overlaid.

The MASTER segment calls subroutines S1, S2, S3 in that order.

In addition subroutine S3 calls functions S4 and subroutines S5, S6 in that order.

It is assumed that the MASTER segment and each subroutine are of roughly the same length and only enough core store is available for the MASTER segment and two subroutines.

In this case the program could be organised as follows:

```
LIST
SEND TO (COMPFILE.SUBFA)
DUMP ON (PROGRAM FILE)
OVERLAY PROGRAM (XMPL)
OVERLAY (1,1) S1
OVERLAY (1,2) S2
OVERLAY (1,3) S3
OVERLAY (2,1) S4
OVERLAY (2,2) S5
OVERLAY (2,3) S6
OUTPUT 1 = LP0
USE 2 = MT1
INPUT 3 = TR0
END
```

Chapter 9 Miscellaneous features

EXTENDED DATA AND COMPACT DATA

1900 FORTRAN may be used on medium and large processors in the 1900 Series. Medium processors are defined as having not more than 32768 words of core store; large processors normally have at least 65536 words of core store. Object programs that are to utilise fully the core stores of large processors must be *extended data* programs; these programs cannot be run on medium processors. Other programs are known as *compact data* programs; compact data programs may be run on medium or large processors; in the latter case the program area may not occupy more than 32768 words of core store.

A program that is to be compiled and run on a large processor may contain any amount of data, subject to the limitations of the core store, in the form of arrays or COMMON variables. The core store required for holding other parts of the program must not exceed 32768 words; no individual COMMON block may be longer than 32768 words (i.e. not more than 16384 REAL variables).

Compiling and running on different size processors

The disc compiler will normally produce a compact data program on a medium processor and an extended data program on a large processor. The following compiling system statements allow for running or re-compiling on different sized processors.

EXTENDED DATA

COMPACT DATA

MIXED SEGMENTS

If a program is compiled on a medium processor for running on a large processor it must contain the EXTENDED DATA statement in the program description. However, this statement may be omitted if the program will not occupy more than 32768 words of core store.

If a program is compiled on a large processor for running on a medium processor it must contain the COMPACT DATA statement in the program description.

Mixed language programs

This section provides additional information needed only by users wishing to incorporate PLAN subroutines into a FORTRAN program or otherwise to mix languages. (This includes reading from other libraries.) Chapter 11 describes mixed language programming in detail.

Compilers mark individual semi-compiled segments to indicate that they can either run only in a compact data program, run only in an extended data program or run in either type of program. Those segments which can be run in either type of program are marked as 'either' by the respective compiler.

FORTRAN, Algol and E.M.A. produce segments marked as capable of running in either type of program. For a PLAN segment it is up to the programmer whether or not the segment will work in compact data programs or extended data programs. Segments produced by all PLAN 1, 2 and 3 compilers are marked as compact data segments. PLAN 4 compilers allow the user to specify how the segments should be marked.

The disc consolidator will check the mode of all semi-compiled segments against the mode of the program. Segments marked 'either' will be accepted under the usual conditions. Compact data segments in an extended data program or extended data segments in a compact data program will normally be faulted; however, if the segments appear in a subroutine library they will be ignored. It is thus possible to have alternative versions of the same subroutine for working in extended or compact data programs.

The programmer can override the checks described above by including the statement

MIXED SEGMENTS

in the program description. Segments are then accepted regardless of their mode. The main reason for including this statement is to allow segments which will work in extended data programs to be included even though they are marked as compact data only.

PRIORITY

Some of the larger machines in the 1900 Series are multi-programming; that is, several programs can be in core store at any one time. When the execution of one program is suspended whilst a peripheral transfer is taking place, Executive will continue with another program. This program may in turn be suspended and then control will be passed to a third program and so on. This system permits the simultaneous use of a large number of peripherals at maximum efficiency, as well as permitting the most effective use of the central processor. To facilitate the organisation of time sharing between programs, each program is given a priority, which indicates the ratio of peripheral activity to actual computing time. A program with high peripheral activity comparing to actual computing time requires a high priority and vice versa. A priority is assigned to a program by including in the program description a statement of the form

PRIORITY *n*

where *n* is an integer in the range 1 to 99. If this statement is omitted, a priority of 50 is assumed. The statement will have no effect on a single programming machine.

THE OMIT STATEMENT

If a program is under test, some segments may not be required when the object program is run; some may not even be written. Segments not required can be omitted at the consolidation phase by including the following statement in the program description:

OMIT *seg 1, seg 2, . . . , seg n*

seg 1, seg 2, . . . , seg n is a list of the names of segments not to be consolidated into the object program. The segments may or may not be presented to the compiler. The program will be terminated if a call to a missing segment is encountered at run time.

THE COMPRESS STATEMENTS

Integer and logical statement

This statement may be written anywhere in the program description as follows:

COMPRESS INTEGER AND LOGICAL

It causes one word of core store instead of two to be assigned to each INTEGER and LOGICAL variable or array element used in the source program. The range of INTEGER quantities is not affected, and any operations on INTEGER quantities will yield the same results. Two words are used normally only for compatibility with other versions of FORTRAN. The user is normally recommended to use this statement only when store shortage makes it essential or when some segments in a program are written in another language and these segments require compressed FORTRAN INTEGER quantities.

In COMMON, EQUIVALANCE and DATA statements it is advisable to match items of the same type; otherwise erroneous results may be obtained when a COMPRESS statement is used. For example, in one segment the statement

COMMON/MAT/A,B(100), I(10,10), C

appears and in another

COMMON/MAT/ING,BB(100), J(10,10), CC

If both segments are compiled without the COMPRESS statement, ING and A will occupy the same storage; as will B and BB, I and J, C and CC, whereas if both segments are compiled in COMPRESS INTEGER AND LOGICAL mode, the variables and arrays following A and ING will no longer correspond. If the common area is being used to pass values from one segment to the other, incorrect results will be obtained.

Semi-compiled segments presented to the compiler are not affected by COMPRESS statements.

A side effect of the COMPRESS INTEGER AND LOGICAL mode of compilation is that all TEXT constants used in the program are rounded up by the addition of spaces to the next multiple of four characters, instead of eight, even if TEXT quantities are to be held in REAL, DOUBLE PRECISION, or COMPLEX variables and array elements.

Double precision

The statement

COMPRESS DOUBLE PRECISION

included anywhere in the program description, causes all DOUBLE PRECISION variables and array elements used in the source program to be treated as type REAL. This limits the accuracy but does not change the range of DOUBLE PRECISION variables and array elements. When a compressed DOUBLE PRECISION variable or array element is to contain TEXT information, the maximum number of characters that can be stored is 8 instead of 16. Care should be exercised when using COMMON, EQUIVALENCE, and DATA statements.

Functions that require DOUBLE PRECISION arguments may still be used but the accuracy obtained may be slightly less than that obtained for a REAL variable. This is because two arbitrary words will be taken to lengthen each argument to the required four words. The inaccuracy caused will be so slight as to be insignificant in the majority of cases.

DOUBLE PRECISION working uses more core store and takes considerably longer than REAL working, even on machines with floating point hardware. If data is not accurate enough to justify DOUBLE PRECISION working, or if highly accurate results are not required, the COMPRESS statement can be used to reduce run time in a program that normally requires DOUBLE PRECISION working. Similarly, when a program is under test, the COMPRESS statement can be used to reduce run time and subsequently be removed for production runs.

This statement is of use when programs written for other machines and using DOUBLE PRECISION are to be run on 1900 Series machines, and the accuracy of the REAL working of the 1900 is found to be adequate. The COMPRESS statement is inserted once, and saves changing each type statement.

Chapter 10 Editing

DESCRIPTION

General

The disc editor #XMED provides a simple system for creating and editing FORTRAN source files on disc, which can then be input to the FORTRAN disc compiler. #XMED is first used to copy the source programs from cards or paper tape to a disc file, and each program or set of data is inserted into an individual subfile. It can then be used to read the subfiles from this disc file and produce an amended copy on a second disc file. Source or data subfiles may be amended, deleted or inserted, and such editing is by line number within the subfile.

The new source program or data lines or subfiles are input from cards or paper tape; they comprise the *editing file*. Line printer listings and card or paper tape copies of the source programs or data may also be produced.

Input

Input to #XMED consists of:

- 1 The editing file, containing the program parameters and information to be used to create or amend subfiles.
- 2 On an updating or copying run, the disc file to be updated or copied. This must conform to the 1900 Series standard file formats which are described in the manual *Compiling Systems*.

Disc files output by #XMUM can be edited by #XMED.

#XMED can create or update files that contain different jobs. A job can consist of:

- 1 A FORTRAN source program.
- 2 GEORGE 1 and 2 macros.
- 3 Data consisting of characters in the 64 character printing set. A data record containing more than 112 characters will be processed in batches of 112 characters.

Both 1900 Series and Atlas card codes can be used for input to #XMED. All the disc files used must have 128-word buckets and two-word bucket headers.

Output

The direct access file produced by #XMED is a standard 1900 Series composite file, consisting of level 0 subfiles. The size of the file depends on the amount of data the file is to hold, and can be calculated by assuming that 504 characters of data will fill one bucket. #XMED prints the size of each file after each run.

ERROR DETECTION

#XMED detects a wide variety of error situations and outputs appropriate messages to indicate the action taken. In particular, #XMED will not attempt to extend a file should it become full; however, the incomplete source program may still be input to the compiler or re-input to #XMED, if desired.

LISTING OF A PROGRAM

When a subfile is created or amended, the latest version of the program is listed automatically on the line printer. The lines are numbered from 1, insertion lines being marked by a * character. If a record on the disc file contains more data than can be listed on the line printer, a > character will appear between the line number and the line itself and the remaining lines are not listed. The complete record is, however, output to the second disc file.

Where lines are deleted, the message:

n LINES DELETED

appears.

For source programs on cards, sequence errors are marked by:

SEQ ERR

at the end of the line; such cards are not rejected by the program.

Any subfiles not referenced during an amendment run will be copied across to the output file unchanged, but will not be listed.

A file summary is printed at the end of the run listing the names of the subfiles on the output file and their lengths in buckets. Source subfiles that have been explicitly referenced are indicated by a * character in front of their names.

Listing takes place automatically when a program is being created or amended, but not if it is being deleted or copied unchanged. To suppress or force listing of a program, see *NOLIST, page 53 and *LIST, page 54.

DIRECTIVES

The various editing operations to be carried out are specified by statements known as *directives* input to #XMED on cards or paper tape. All the editing directives are described later in the Chapter. Details of the operating instructions for #XMED may be found in Chapter 14, page 113.

Each directive occupies one line or card. The first character of any directive is generally a *, and always appears in the first position of a line, or in column 1 of a card. Subsequent characters follow immediately: they are alphabetic and the first three only are significant. The only directives not including alphabetic characters are the subfile terminating directive **** and the batch terminating directive ///.

Some directives have parameters associated with them. In general, if parameters are present they should be separated from the directive and from each other by at least one space, ∇, or tabulation character, as appropriate. However, if a parameter starts with a (character it need not be separated from whatever precedes it, and certain other characters (such as , and =) obviate the need for space characters between parameters. Within the parameters spaces and/or tabulation characters may be present, except within numbers. For file names, spaces are significant.

The order of card or paper tape input (the editing file) is:

- 1 Batch directives which specify the files to be opened and control the operation of #XMED
- 2 The subfile directive which specifies the subfile to be processed and the source language or data in the subfile
- 3 Other directives relating to the subfile (optional)
- 4 Source program or data lines or amendments
- 5 The subfile terminator ****
- 6 The batch terminator ///

In order to create or amend more than one subfile in a file during a single #XMED run, it is only necessary to repeat these for each subfile.

Blank lines between directives are ignored, but blank lines within a program are written to the file.

Batch Directives

The first directives input to the program define the direct access file(s) to be used. They may appear in any order, but must all precede the first subfile directive.

*IN

Format: *IN∇x(f(g))

This directive specifies the disc file used for input to #XMED. x indicates the device type for the file and is either ED for an E.D.S. file or FD for an F.D.S. file. If x is omitted, ED is assumed. f is the input file name and g its generation number. If g and its enclosing parentheses are omitted, the file named f with the highest generation number in the system is assumed. The input file is normally a file created by a previous run of #XMED which is to be updated. The *IN directive should not be present when a disc file is being created.

***OUT**

Format: ***OUT** $\forall x(f(g))$
 or
 ***OUT** $\forall x(f(g) = f'(g'))$

The ***OUT** directive specifies the file to be opened as an output file, to which newly created or amended programs are written.

For the first form, x is the output device type, f the file name and g its generation number. x and g may be omitted, as for ***IN**.

The second form is used if it is desired to use the file with name f and generation number g for output but to change the name to f' and the generation number to g' . g' and its enclosing parentheses may be omitted, in which case a new generation number of zero is assumed.

The ***OUT** directive should always be present except when only a listing or card or paper tape copy of an input file is required, see *Job directives*, below.

***LOAD**

Format: ***LOAD** $\forall \#name, \#file$

This directive is optional and, if present, will cause **#XMED** to delete itself at the end of the run and load the program called $\#name$ from the file called **PROGRAM** $\forall file$. If $\#name$ and $\#file$ are exactly the same then $\#file$ and its preceding comma may be omitted.

The batch terminator

Format: **////**

This directive must always terminate the editing file, and must be followed by a blank card or an extra newline character.

The subfile directive

The subfile directive is the first directive for each source program or set of data and defines the language of the subfile. Formats of immediate interest are:

*FORTRAN $\forall s, u$	for a FORTRAN job
*GEORGE $\forall s, u$	for a set of GEORGE macros (see Chapter 12, page 73)
*DATA $\forall s, u$	for a set of data

In each case, s is the subfile name which must always be specified and which will consist of up to 12 alphanumeric characters, spaces or hyphens, the first character being alphabetic. If the subfile name is omitted, the directive and any following source data or amendments will be ignored. u is a user identification which may consist of any characters the user wishes or may be omitted, together with the preceding comma. Any user identification included will be ignored by **#XMED** and will not be written to the direct access file.

If a subfile directive refers to a subfile that is not already in the file, a new subfile of the given name and type will be inserted at the end of the file.

The subfile terminator

Format: ********

This directive terminates the source program, data or amendment lines for each subfile.

Job directives

The following directives, which can appear in any order and are all optional, apply only to the subfile specified in the subfile directive immediately preceding them. They must appear before the first line of source program or data, or before the first ***ALTER** directive in an amendment run.

***NOLIST**

Format: ***NOLIST**

The ***NOLIST** directive is used to suppress the listing of the program. It is used when a subfile is being created or amended and a listing is not required. It is otherwise redundant, the listing being suppressed automatically.

***LIST**

Format: ***LIST**

This directive causes listing of the program or data in its latest form. It is used when a listing is required, but when the source file is not being created or amended; that is, when there is no ***OUT** directive, or when a file is being copied without being amended. It is otherwise redundant, since the listing is produced automatically.

***CPCOPY**

Format: ***CPCOPY**

The ***CPCOPY** directive causes a card copy of the subfile in its latest form to be produced. The copy will always be in 1900 code.

***TPCOPY**

Format: ***TPCOPY**

This directive causes a paper tape copy of the subfile in its latest form to be produced.

***DELETE**

Format: ***DELETE**

The ***DELETE** directive causes the specified subfile to be deleted. It may be used only where there are both input and output files and it is not possible to obtain a listing or a card or paper tape copy of the deleted subfile.

***LAST**

Format: ***LAST***abcd*

The ***LAST** directive permits the use of a four character subfile terminator directive other than the normal ********. *abcd* can be any four printable characters other than spaces, but must not begin with any of the four characters **\$] ↑ or ←** and must not be the same as the first four characters of any of the **#XMED** directives.

Other directives

***ATLAS, *1900**

Format: ***ATLAS**
 ***1900**

#XMED will initially assume either Atlas or 1900 card code, depending on the entry point used (see Operating instruction 2, page 115). If there is a change of card code anywhere in the input stream, these two directives are available for informing **#XMED** of the change. Each indicates the card code changed to.

They may appear anywhere in or before the source program or data. At the beginning of each subfile the card code specified by the entry point is assumed. The paper tape entry point is for input in 1900 Series code.

***SWITCH**

Format: ***SWITCH**

When the input to **#XMED** is on both cards and paper tape, the ***SWITCH** directive is used to make a switch from card reader to paper tape reader or vice versa. The directive may appear anywhere in the editing file. It must always be followed by a blank card or line which is read, but ignored, by **#XMED**. If the entry point to **#XMED** indicated paper tape input, the ***SWITCH** directive will cause **#XMED** to expect cards in 1900 code. After a peripheral has been ***SWITCHed** to, it remains on-line until another ***SWITCH** is encountered, irrespective of change of subfile.

***WARN**

Format: ***WARN***∇c*

Use of this directive permits the editing of files containing records that are similar to standard **#XMED** directives. It stipulates that the first character, or *warning character*, of all subsequent **#XMED** directives is to be changed

from * to *c*, where *c* can be any character from the ICL 1900 Series 64-character set, except a space character or one of the four characters \$] ↑ or ←. The *WARN directive can appear anywhere in the editing file, and the new warning character will remain in force until the end of the run, unless it is changed by a cWARN directive. (The *WARN directive is implemented as soon as it is read and therefore all subsequent directives, including the cWARN directive, must commence with the new warning character.) If the new warning character is still in force, the batch terminator directive must be written *c*///. There is one exception to this rule in that the special subfile terminator specified by a *LAST directive is unchanged by *WARN.

A cWARN directive without subsequent parameter reinstates an asterisk as the warning character and /// as the batch terminator.

In the absence of a *WARN directive, and in the case of a second or subsequent run (unless a *WARN directive is used in those runs) the warning character * and batch terminator /// are assumed.

*ALTER

The line editing directive *ALTER indicates that source lines are to be deleted, inserted or amended and is only acceptable if both an input file and an output file are specified. Lines are referenced by line numbers, which are provided by #XMED when a listing of the program is produced.

Deletion and amendment of lines

Format 1: *ALTER*∇**l-m*

Format 2: *ALTER*∇**l, n*

When existing lines are to be deleted and possibly replaced by amendment lines, either of the above formats may be used.

l is the line number of the first line to be deleted.

m is the line number of the last line to be deleted.

n is the number of lines to be deleted.

The directive will be followed, if required, by any amendment lines which are to replace the deleted lines. All source lines (i.e. non-directive lines) following the *ALTER will be inserted until the next *ALTER directive or the subfile terminator is encountered.

Insertion of lines

Format: *ALTER*∇**p*

This format is used when lines are to be inserted, but no deletions are to be made. *p* is the line number of the line before which the new lines are to be inserted. If *p* is one greater than the line number of the last line of the program, the new lines will be inserted at the end of the program.

The directive will, as before, be followed by the new lines to be inserted.

*Order of *ALTER directives*

*ALTER directives should appear in ascending order of line number, and the line number of a line previously deleted must not be used.

Examples of input to #XMED

- 1 A FORTRAN source program is on cards and it is required to copy it on to a disc file. Input to #XMED consists of the following cards:

OUT∇*(DISCFILE(1))

FORTRAN∇*SOURCE,CD7

FORTRAN source cards

///

Blank card

The *OUT directive specifies the output file to be an E.D.S. file DISCFIL with generation number 1. The source program will be held in a subfile of DISCFIL called SOURCE.

- 2 It may be necessary to make some alterations to the source program and to do this a second E.D.S. file will be required. The input on card or paper tape could be as follows:

```
*INV(DISCFIL(1))
*OUTV(DISCFIL(2))
*FORTRANVSOURCE,CD8
*ALTERV2,1
      DIMENSION A(5,5),B(10,10),C(15,15)
      COMMON/SHARE/A,B,C
*ALTERV22,1
      DIMENSION F(5,5),G(10,10),H(15,15)
      COMMON/SHARE/F,G,H
*ALTERV30
      PAUSE 55
*ALTERV50-58
****
////
Blank card or line
```

The *IN directive specifies the file created previously and the *OUT directive specifies that the output file is to be an E.D.S. file called DISCFIL with generation number 2.

The *ALTER directives specify the alterations to be made. Lines 2 and 22 of the program are each to be replaced with a DIMENSION and a COMMON statement. A PAUSE statement is to be inserted immediately preceding line 30, and lines 50 to 58 are to be deleted. The line numbers were listed by #XMED on the previous run.

DISCFIL(2) could now be used as input to the next editing run.

- 3 If an extra listing and card copy of the FORTRAN program in DISCFIL(2) are required, the card or paper tape input would be as follows:

```
*INVDISCFIL(2)
*FORTRANVSOURCE,CD8
*LIST
*CPCOPY
****
////
Blank card or line
```

Chapter 11 Mixed language programming

A mixed language program may be created by consolidating together a number of semi-compiled segments. Nearly all 1900 compilers output segments in this form and thus FORTRAN segments may be mixed with PLAN segments, Algol segments etc. This chapter is concerned mainly with the inclusion of PLAN segments in a FORTRAN program.

COMMUNICATION OF DATA

The communication of data between PLAN and FORTRAN segments may be achieved either by passing over arguments or via a COMMON area. Table 4, page 66 gives details of the number of words used to store FORTRAN variables.

TRANSFER OF CONTROL

The transfer of control to a segment in the other language is effected by the execution of a *calling sequence* within the calling segment. Control is then passed to the called segment, which will, normally, carry out the following:

- 1 Preliminary operations such as storing the link address to which control is to be returned on completion of the segment.
- 2 The main body of operations.
- 3 Terminal operation – returning to the calling segment via the link.

CALLING PLAN SEGMENTS FROM FORTRAN

A PLAN segment may be called from a FORTRAN segment either as a SUBROUTINE segment or a FUNCTION segment, the difference being that a SUBROUTINE segment may have any number of arguments, including zero, and a FUNCTION segment must have at least one argument. Furthermore a FUNCTION segment must return a result in a location that depends on the type of function. Table 3 lists the location of results for different types of function.

Type of function	Location of result
Integer or Logical	X6
Real	Floating-point accumulator
Double-precision	Floating-point accumulator: first two words of result X4,X5: Second two words of result.
Complex	Floating-point accumulator: real part of result. X4,X5: imaginary part of result.

Table 3 Location of results for different functions

Both a SUBROUTINE segment and a FUNCTION segment may return results via a common area or via the arguments.

When writing FUNCTION or SUBROUTINE segments in PLAN, programmers are advised to use the subroutines FPROLOG and FEPILOG, which are included in the FORTRAN library. The function of FPROLOG is to pick up and store the link and the address of each argument in an area of store specified by the programmer; the function of FEPILOG is to execute the return of control to the calling segment.

Using these subroutines, a PLAN segment will consist of the following:

CALL to FPROLOG

Main body of segment

CALL to FEPILOG

and should start with a #PROGRAM directive of the form:

#PROGRAM /name

where *name* is the name of the segment as in the CALL statement or function reference.

The FPROLOG subroutine

FUNCTION

To provide communication between a FORTRAN calling segment and a PLAN segment.

DESCRIPTION

This subroutine should be called by PLAN segments that are to be incorporated into a FORTRAN object program as functions or subroutines. The subroutine transfers a fixed or variable number of arguments from a FORTRAN calling sequence to a storage area specified in the PLAN segment, and passes the name of the called segment to the FORTRAN trace system.

The first word of the storage area specified in the PLAN segment is used to store link information for the return to the FORTRAN segment. This word must not be altered in any way during the segment. The return must be effected by a call to the FEPILOG subroutine. (See *The FEPILOG subroutine*, page 59)

No instructions should precede the call to FPROLOG in the segment.

INPUT ARGUMENTS

The PLAN segments should commence with the following calling sequence:

CALL 3 FPROLOG

8Hname

c

/addr

name is the name of the PLAN segment as it is to appear in any FORTRAN trace output. This name is held in two words, which should be space-filled if necessary, e.g. 8HPSEGVVVV.

For PLAN segments requiring a *fixed* number of arguments, $c = n$, the number of arguments to be transferred. For PLAN segments requiring a *variable* number of arguments $c = -m$, where m is the maximum number of arguments to be transferred, excluding the first. The first argument must specify n , the number of actual arguments for the particular call to the segment. If n is found to exceed m then a fatal execution error is signalled.

addr is the start address of the area, in lower or upper data, that is to receive the link information and arguments. This area must contain $n + 1$ words if n is fixed or $m + 2$ words if n is variable.

When the PLAN segment is to deal with a variable number of arguments, the first argument in the FORTRAN argument list must be an additional argument specifying the number of subsequent arguments; this first argument must be of type INTEGER and it may be a constant, a variable or an array element.

RESULTS

The result of a call to FPROLOG is as follows:

For a fixed number of arguments:

addr contains link information for use by FEPILOG.

addr + 1 contains the address of the first argument.

addr + 2 contains the address of the second argument.

.....

addr + n contains the address of the n th argument.

For a variable number of arguments:

addr contains link information for use by FEPILOG.

addr + 1 contains *n*, the value of the first argument (giving the number of arguments to follow).

addr + 2 contains the address of the second argument.

...

addr+*N*+1 contains the address of the last argument.

USE OF OVERFLOW

If overflow is found to be set when FPROLOG is called then FORTRAN execution error 50 will be reported.

SPECIAL POINTS

- 1 No instructions should be obeyed before the call to FPROLOG.
- 2 The address of the argument as given by FPROLOG is contained in the modifier part of the word; the remaining bits may be non-zero.
- 3 The word containing the link address should not be altered in any way.

The FEPILOG subroutine

FUNCTION

To return control from a PLAN SUBROUTINE or FUNCTION segment to the FORTRAN calling segment.

DESCRIPTION

This subroutine utilises link information generated by the subroutine FPROLOG, to effect a return from a PLAN segment to the FORTRAN calling segment; in addition the subroutine passes information to the FORTRAN trace system. The return of control is made directly from FEPILOG to the FORTRAN segment so that no EXIT instruction is required in the PLAN segment.

For every call to FPROLOG, within a program, there must be, dynamically, a corresponding call to FEPILOG.

INPUT ARGUMENTS

The calling sequence is as follows:

CALL 3 FEPILOG

/addr

addr is the address of the location containing the link information stored by the corresponding call to FPROLOG.

RESULTS

Control will be returned to the calling segment and this will be noted by the FORTRAN trace system.

USE OF OVERFLOW

If overflow has been set during the PLAN segment and is still set on the calling of FEPILOG then FORTRAN execution error 50 is signalled.

Example

A FUNCTION segment is to be written for the purpose of adding together two complex numbers. The segment could be written in PLAN as follows:

LABEL	OPERATION	ACC.	↑ OPERAND													
1	6	7	12	13	15	16	20	24	28	32	36	40	44	48	52	56
#PROGRAM						/CADD										
#LOWER																
						LINK(3)										
#PROGRAM																
	CALL	3				FPROLOG										
						8HCADDVVVV										
						2										
						/LINK										
	LDX	1				LINK+1										
	LDX	2				LINK+2										
	LFP					2(1)										
	FAD					2(2)										
	SFP					4										
	LFP					0(1)										
	FAD					0(2)										
	CALL	3				FPEILOG										
						/LINK										
#END																

This segment would be equivalent to the FORTRAN segment:

```

COMPLEX FUNCTION CADD (A,B)
COMPLEX A,B
CADD = A+B
RETURN
END

```

Arrays

In a FORTRAN program arrays are stored with their elements in consecutive storage units, which may consist of one, two or four words of storage, so that the leftmost subscript varies most rapidly and successive subscripts vary correspondingly less rapidly. In order that complete arrays may be 'handled' easily, information about each array is stored separately in an *array header*.

Two PLAN subroutines, GETAH and GENAH, are available for dealing with the array headers of FORTRAN arrays that are passed over into PLAN segments.

THE GETAH SUBROUTINE

Function

The subroutine unpacks a FORTRAN array header into an information block accessible to a PLAN segment.

Input arguments

The calling sequence is as follows:

```
CALL 1 GETAH
```

```
LDX 3 a
```

```
LDX 3 b
```

a contains the address of the array header.

b contains the address of the first word of the area of store that is to receive the information block. For an *n*-dimensional array the length of the information block must be *n*+4 words.

Results

The results returned to the information block are as follows:

WORD 0 contains the number of dimensions of the array (*n*).

WORD 1 contains the number of words per array element (*m*).

WORD 2 contains the Base address – the address of elements (0,0,0,...0).

WORD 3 contains the address of the first element of the array – element (1,1,1,...,1).

WORD 4 contains the address of the first word past the end of the array.

WORD 5 contains the first partial product P_1 (only present for $n > 2$).

WORD 6 contains the second partial product P_2 (only present for $n > 3$).

⋮

WORD (*n*+3) contains the last partial product P_{n-1}

Notes

P_k is defined by $P_k = D_1.D_2.D_3.....D_k$

where D_k is the number of elements in dimension *k*, and *k* may take any of the values 1,2,3,...,*n*-1. The base address is found by subtracting *m*.*N* from the address of the first element, where $N = 1 + P_1 + P_2 + ... + P_{n-1}$

THE GENAH SUBROUTINE

Function

The subroutine generates a standard FORTRAN array header from an information block as produced by GETAH.

Input arguments

The calling sequence is as follows:

```
CALL 1 GENAH
```

```
LDX 3 a
```

```
LDX 3 b
```

b contains the address of the first word of the information block and *a* contains the address of the first word of the area at which the array header is to be generated. If *a* and *b* are specified as the same, the effect will be to condense the information block into a standard array header.

It is recommended that the same number of words are set aside for both the information block and the array header.

Results

A standard FORTRAN array header is generated at the address specified.

EXAMPLE

It is desired to write a PLAN subroutine, to be called by a FORTRAN segment, to copy one array into another.

The arrays are assumed to have a similar structure and not more than three dimensions. Two arguments, i.e. two array names, are required by the subroutine and the corresponding information blocks are called IB1 and IB2.

A possible version of the segment is given on pages 62 and 63.

ICL 1900 SERIES						TITLE																		SEGMENT			
PLAN CODING SHEET						PROGRAMMER																		DATE			
LABEL	OPERATION	ACC.	↑ OPERAND ↑																		PROG. IDENT.	SEQUENCE					
1	6	7	12	13	15	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	73	75	76	80			
#PROGRAM							/COPYARRY																				
#LOWER							LINK(3), IB1(7), IB2(7)																				
#PROGRAM																											
	CALL	3					FPROLOG																				
							8HCOPYARRY																				
							2																				
							/LINK																				
	CALL	1					GETAH																				
	LDX	3					LINK+1																				
	LDN	3					IB1																				
	CALL	1					GETAH																				
	LDX	3					LINK+2																				
	LDN	3					IB2																				
	LDX	3					IB1+4																				
	SBX	3					IB1+3																				
	LDX	7					IB2+4																				
	SBX	7					IB2+3																				

SHEET 1 OF 2

[illegible]

Common storage areas

In the previous examples information has been passed over to the called segment via arguments. An alternative, or complementary, method is to pass information via a common storage area.

When establishing common areas care must be taken to ensure that the differences in the sizes of the storage units used by FORTRAN and PLAN are allowed for.

Details of the number of words used to store FORTRAN variables are given in Table 4, page 66. When an array is specified in a FORTRAN COMMON statement, the number of words occupied by the array is the product of the number of elements and the number of words per element.

It must be remembered that when arrays are being dimensioned, the number enclosed in parentheses in FORTRAN refers to the number of elements, whereas in PLAN the number enclosed in parentheses refers to the number of words.

Thus the following FORTRAN statements:

REAL B

COMMON/AREA/B(7)

and the PLAN directive:

LABEL	OPERATION	ACC.	↑ OPERAND											
1	6	7	12	13	15	16	20	24	28	32	36	40	44	56
#UPPER						COMMON/AREA/								
						C(14)								

establish a common area named AREA in which B corresponds to C and both occupy 14 words. The PLAN directive #LOWER could be used in place of #UPPER to force the area to be placed in lower storage, although the FORTRAN segment(s) could not take advantage of this.

EXAMPLE OF THE USE OF COMMON

A FORTRAN segment contains the following statement:

LOGICAL BOOLEAN

COMMON/POOL/J,A(5), LIST (5,2), BOOLEAN

These establish a common area of 34 words called POOL, as follows:

INTEGER variable J 2 words

REAL array A 10 words

INTEGER array LIST 20 words

LOGICAL variable BOOLEAN 2 words

A PLAN segment called by the FORTRAN segment contains the directive:

LABEL	OPERATION	ACC.	↑ OPERAND											
1	6	7	12	13	15	16	20	24	28	32	36	40	44	56
#UPPER						COMMON/POOL/								
						ITEM, NULL, RES(10), KI(10), K2(10), Y(3)								

This establishes a common area of 35 words called POOL as follows:

ITEM	1 word
NULL	1 word
RES	10 words
K1	10 words
K2	10 words
Y	3 words

During consolidation the two areas will be combined into one storage area named POOL consisting of 35 words (the larger of the sizes) of which the FORTRAN segment may use only the first 34 words as follows:

J	will correspond to ITEM and NULL.
A	will correspond to RES.
LIST	will correspond to K1 and K2.
BOOLEAN	will correspond to the first two words of Y.

BLOCK DATA SEGMENTS

The equivalent of a FORTRAN BLOCK DATA segment may be written in PLAN. The initial #PROGRAM directive must be of the form:

#PROGRAM *programe/segname*

where *programe* is the program name and *segname* is any unique segment name.

The segment must contain only #UPPER or #LOWER COMMON directives that set up common areas containing preset values.

BLANK COMMON AREAS

The blank common area may be used by both PLAN and FORTRAN segments, for example the blank common block established by the statement:

COMMON FRED, BERT

could be accessed by a PLAN segment containing the directive:

LABEL		OPERATION		ACC.		OPERAND										
1	6	7	12	13	15	16	20	24	28	32	36	40	44	48	52	56
#UPPER						COMMON//										
						JIN(4)										

<i>Type</i>	<i>Number of words</i>	<i>Format</i>
<i>NORMAL COMPILATION MODE</i> REAL	2	Word 1 bits 0 to 23 signed mantissa Word 2 bit 0 0 bits 1 to 14 continuation of mantissa bits 15 to 23 exponent + 256 (i.e. as PLAN floating point number)
INTEGER	2	Word 1 bits 0 to 23 signed binary integer Word 2 bits 0 to 23 unused
LOGICAL	2	Word 1 bits 0 to 22 0 bit 23 0 for .FALSE. 1 for .TRUE. Word 2 bits 0 to 23 unused
DOUBLE PRECISION	4	Word 1 } as for REAL Word 2 } Word 3 bit 0 0 bits 1 to 23 continuation of mantissa Word 4 bit 0 0 bits 1 to 9 final part of mantissa
COMPLEX	4	Word 1 } as for REAL Word 2 } Word 3 } as for REAL Word 4 } The first pair give the real part and the second pair the imaginary part of the complex number.
<i>COMPRESS INTEGER AND LOGICAL MODE</i> INTEGER	1	Word 1 bits 0 to 23 signed binary integer
LOGICAL	1	Word 1 bits 0 to 22 0 bit 23 0 for .FALSE. 1 for .TRUE.
<i>COMPRESS DOUBLE PRECISION MODE</i> DOUBLE PRECISION	2	Word 1 } as for REAL Word 2 }

Table 4 The method of storage of FORTRAN variables

CALLING FORTRAN SEGMENTS FROM PLAN

The following calling sequence should be used wherever a FORTRAN segment is called from a PLAN segment.

CALL 1 *segname*

LDN 3 a_1

LDN 3 a_2

. . .

. . .

LDN 3 a_n

$a_1, a_2, \dots, a_n$ are locations in the LOWER data store containing each argument in turn.

If any of the arguments are held in the UPPER data store then the

LDN 3 a

instruction should be replaced by an instruction of the form

LDX 3 b

where b is the word in the #LOWER data store containing the address of the argument. Both forms are such that when obeyed, the address of the argument is placed in X3. Control is returned to the word following the last LDN or LDX instruction. Table 5 gives the values of the operand and associated instructions for the various types of arguments.

<i>Type of argument required by FORTRAN segment</i>	<i>Instruction</i>	<i>Operand</i>
Constant or variable	LDN 3	Address of constant or variable
	LDX 3	Address or word containing address of constant or variable
Array	LDN 3	Address of array header
	LDX 3	Address of word containing address of array header
	LDX 3	Address of word containing 128/address of first element
Function or Subroutine	LDX 3	Address of word containing a branch instruction (BRN) to function or subroutine

Table 5 Value of operand in PLAN instructions when calling a FORTRAN segment from a PLAN segment

Example

The FORTRAN subroutine PIVOT has been written to determine the element of a one dimensional REAL array that contains the lowest value and to return the element subscript PCOL and the lowest value MIN as follows:

```

SUBROUTINE PIVOT (PCOL,A,N,MIN)
  INTEGER PCOL
  REAL MIN
  DIMENSION A(N)
  PCOL = 1
  MIN = A(1)
  DO 100 I = 2,N
    IF (MIN.GT.A(I)) GO TO 100
    MIN = A(I)
    PCOL = I
  100 CONTINUE
  RETURN
  END

```

It is required to call this subroutine from a PLAN segment in order to process the array SHADOW, which has 50 elements and thus occupies 100 words. The calling sequence might be written as follows:

Label	Operation	Acc.	Operand
1	6 7	12 13 15 16	20 24 28 32 36 40
#UPPER			SHADOW(100)
#PROGRAM			
	LDN	1	50
	STO	1	LENGTH
	CALL	1	PIVOT
	LDN	3	PCOL
	LDX	3	'128/SHADOW'
	LDN	3	LENGTH
	LDN	3	MIN

MIXED PLAN/FORTRAN OVERLAID PROGRAMS

PLAN segments included in a FORTRAN program can be overlaid in the same way as FORTRAN segments, provided they are mentioned in the OVERLAY statement in the program description and are entered via the subroutine FOVER (see below). A PLAN steering segment specifying overlay organisation is not required and must not be input to the compiler. If the PLAN segment itself calls other segments it is advisable that this is done with FOVER.

The subroutine FOVER

FUNCTION

To call FORTRAN overlay system in PLAN segment in a mixed language overlaid program.

DESCRIPTION

In a FORTRAN/PLAN mixed language overlaid program that uses the FORTRAN overlay system, calls from a PLAN segment must normally be achieved by calling FOVER. (For the exceptional case in which this is only optional, see note 1, page 70.)

The function of this subroutine is to ensure that the called segment is in core store (i.e. bringing in the overlay unit containing the called segment, if necessary), and then to exit to the segment (or cue; see note 2, page 70) specified. The normal RETURN statement (or call to FEPILOG) in the called segment causes the overlay system to transfer control to the instruction following the calling sequence for FOVER, after ensuring that it is in core store (i.e. bringing in the overlay unit that contains the instruction if it is an overlay area that has been overwritten since FOVER was called).

If the logic of the program causes control to pass more than once to the section of the program that contains a call to FOVER, on the second and subsequent occasions control will pass more quickly to the called segment.

INPUT ARGUMENTS

The calling sequence is as follows:

```
CALL 1 FOVER
ENTER calledseg
LDN 3  $a_1$ 
LDN 3  $a_2$ 
. . .
LDN 3  $a_n$ 
BRING callingseg
```

where *calledseg* is the name of the called segment

$a_1, a_2, \dots, a_n$ are the n arguments of the called segment

callingseg is the name of the calling segment.

This sequence occupies $(n+6)$ words, since ENTER and BRING are PLAN Macro instructions occupying 3 and 2 words respectively. (They are never executed as such but all the information they contain is used by FOVER.)

USE OF OVERFLOW

Unchanged.

RETURN LOCATIONS

On exit from the called segment return (either direct or via the overlay system) is to CALL + (n+5) (where the CALL referred to is that to FOVER).

Notes

- 1 If the calling and the called segments are to be consolidated into the same overlay unit, or both into permanent store, then the use of FOVER is only optional. The advantage of using it in this case is that subsequent re-organisation of the overlay structure can be achieved without having to recompile any of the PLAN segments involved.
- 2 References to *calledseg* could equally be to *calledseg + c*, *cuenname*, *cuenname + c* (where *cuenname* is the name of cue) when the called segment is written in PLAN.

Restrictions

When using the FORTRAN overlay system the following three restrictions should be noted:

- 1 If the called segment and the calling segment are in different units of the same overlay area, then preset area, which is itself overlaid, should not be used for storing argument addresses or the arguments themselves.
- 2 The PLAN instructions BRING, RECAL and ENTER should not be used except as specified with the FOVER subroutine.
- 3 If an EXIT instruction is used, the segment referred to in the link must currently be in store.

Example

A PLAN segment POLAY with no arguments is to be included in area 1 unit 2 of a FORTRAN program. POLAY calls the FORTRAN segment FSEQ. The program description might include the statements

OVERLAY (1,2) POLAY, OTHERSEGS

OVERLAY (1,3) FSEQ, MORESEGS

and the PLAN segment would have the structure

LABEL		OPERATION		ACC.		↑												OPERAND			
1	6	7	12	13	15	16	20	24	28	32	36	40	44	48	52	56					
#PROGRAM						/POLAY															
	CALL	3				FPROLOG															
						8HPOLAY															
						0															
						/LINK															
	CALL	1				FOVER															
	ENTER					FSEQ															
	BRING					POLAY															
	CALL	3				FEPILOG															
						/LINK															
#END																					

ENTRY POINTS AND MASTER SEGMENTS

There must be one and only one master segment in a program. If written in FORTRAN it must start with a MASTER statement. If written in PLAN it must contain entry point 0. In general, use of entry point 9 should be avoided in a PLAN segment. If any of the FORTRAN segments have been compiled in TRACE 2 mode then no entry points should appear in a PLAN segment.

USE OF WORD 30

In a FORTRAN program the following use of bits in word 30 is made:

Bit 0 Used to control statement tracing of any segment compiled in TRACE 2 mode.

Bits 1 to 8 Used by programmer as sense switches.

Bits 9 to 23 Reserved for future use by FORTRAN system.

In a mixed PLAN/FORTRAN program care should be taken to ensure that a bit in word 30 is not used for more than one purpose.

PERIPHERAL USAGE

It is strongly recommended that input or output on a particular peripheral be programmed in one language only. If this is not done, unpredictable effects may occur.

If a PLAN program incorporates FORTRAN segments containing FORTRAN input/output statements, the peripherals used must be defined in a FORTRAN program description in the usual way and the semi-compiled segment produced from the program description (given the name %%%F) must be included in the PLAN program.

The FORTRAN input/output system will not be incorporated into the object program unless either READ or WRITE statements are present or at least one FORTRAN segment was compiled at trace level 2.

Peripherals nominated in a FORTRAN program description are assigned dynamically, the first time they are used via the FORTRAN system.

SUBPROGRAMMING

There are no subprogramming facilities in FORTRAN but subprogramming can be used in the PLAN segment in the normal way.

COMPILING MIXED PLAN/FORTRAN PROGRAMS

There are many ways of compiling a mixed PLAN/FORTRAN program. Two common methods are described below.

PLAN segments incorporated in a FORTRAN program

FIRST STAGE

#XPLX is used to translate PLAN source segments into unconsolidated semi-compiled form in a disc file.

SECOND STAGE

#XFAE is used to translate FORTRAN source segments into unconsolidated semi-compiled form in a disc file. Consolidation may then be carried out automatically. The PLAN segments, specified by a SEMICOMPILED statement in the FORTRAN compiling system statements, are incorporated. Any library routines called in the PLAN segments may be incorporated from a PLAN library if the appropriate LIBRARY statement is present. The FORTRAN library is scanned automatically.

FORTTRAN segments incorporated in a PLAN program

FIRST STAGE

#XFAE is used to translate FORTTRAN source segments into unconsolidated semi-compiled form in a disc file.

SECOND STAGE

#XPLX is used to translate PLAN source segments into unconsolidated semi-compiled form in a disc file.

THIRD STAGE

Consolidation is carried out as a completely separate process. The files containing the unconsolidated semi-compiled segments are specified by parameters. The PLAN and FORTTRAN library blocks must also be scanned at this stage, so the appropriate parameters must be input to the consolidator.

Chapter 12 GEORGE 1 and 2

INTRODUCTION

This chapter provides information on running the FORTRAN 16K disc compiler #XFAE under GEORGE 1 and 2.

A program or sequence of programs run under the control of GEORGE is called a *job*. The set of instructions that the user gives to GEORGE specifying the requirements of his job is called a *job description*. The job description consists of a set of operating instructions, which are punched either on cards or paper tape. The instructions are commands prepared by the user, which specify the action to be taken at certain stages during the running of the job, and in response to certain events. In general, the operator will be active only at the beginning and end of the job, but under certain circumstances GEORGE may request operator intervention.

Job descriptions are introduced by the JOB command, whose format is:

JOB *job name*, *account number*, *user identification*

The parameters must always appear in the order specified.

An example of the use of the JOB command is:

JOB TEST, 77101, BROWN

where TEST, 77101 and BROWN are the three parameters.

A job description is terminated by the standard terminator **** which, unlike other commands, must start in column 1.

There are three ways in which FORTRAN programs may be compiled and run under GEORGE 1 and 2:

- 1 By using standard macros.
- 2 By using the FORTRAN Batch Development System.
- 3 By the user writing his own job description.

Standard ICL macros are designed for use when compiling and running straightforward FORTRAN jobs, such as program testing or production runs. Source programs may be on paper tape, cards, magnetic tape, E.D.S. or F.D.S. Programmers may provide their own program descriptions, or the standard default program descriptions may be used. A full description of the use of standard macros is given in the section *Standard macros*, page 74.

The FORTRAN Batch Development System enables batches of small FORTRAN jobs to be developed. Source is held on a disc or magnetic tape file, where it may be amended. One set of data for each job can also be held on this file, and updated if necessary. The System will edit, compile, and run a batch of jobs; program descriptions may be provided, or the standard default program descriptions used. The System is described in the section *Batch Development System*, page 87.

The macros described in this chapter may be modified as required by using #XMED or #XKYA, comprehensive specifications of which may be found in the manual *Compiling Systems*. In particular, if #XFAE is the most frequently used compiler in a system, the parameter default value associated with the compiler can be changed from XFAT, the existing value, to XFAE.

The programmer may write his own job description, not using standard ICL macros, if he requires more complex facilities than can be provided with a standard macro; for example, steering trace, or the use of basic peripherals in a way not provided for by the standard macros. A full description of this method is given in the section *User job descriptions*, page 105.

Little knowledge of GEORGE 1 and 2 is assumed in the sections *Standard macros* and *Batch Development System*; the user is assumed to be familiar with GEORGE 1 and 2 in the section *User job descriptions*.

STANDARD MACROS

This section describes the standard macros supplied by ICL for use when running the FORTRAN 16K disc compiler, #XFAE, under GEORGE 1 and 2.

Macros available

The following macros are available to the user:

<i>Macro name</i>	<i>Function</i>
FORTTRAN	Compiles, consolidates, and runs a program.
FORTTRANCOMP	Compiles, consolidates, and saves binary output.
FORTTRANCOMPX	Similar to FORTTRANCOMP, but for use when source input is on backing store.
FORTTRANSEMI	Compiles and saves semicompiled output.
FORTTRANRUN	Runs a binary program.

Specifications of these macros, and examples of their use, are given later in the Chapter.

Programming information

Apart from FORTTRANCOMPX, the macros are designed to accept source input from cards or paper tape punched in 1900 code (see Appendix 3, page 131).

A four character identifier must be associated with each program. This is used as the name of the program, in source and data *document names*, and in certain file names. The document names assigned to the source program and data input must be

FORT*ident*

and

DATA*ident*

respectively, where *ident* is the four character program identifier.

If the user does not provide a program description, a standard default program description is incorporated automatically, and both source program and data are assumed to be on the same type of peripheral as the job description. However, a READ FROM statement may be used with any of the macros to switch the medium of the compiler input stream (see Chapter 6, page 25).

All necessary disc files must be either all on E.D.S. or all on F.D.S., except for the source file for FORTTRANCOMPX.

The user must supply his own program description if he requires facilities other than those provided by the standard default program description. A list of program description statements is given in Table 1, page 3.

The succeeding pages contain specifications of the GEORGE 1 and 2 standard macros in alphabetical order. The parameters applicable to each macro are described, and the default program description is given in each case. The specifications also include notes on the documents and disc files required, and any restrictions imposed by the use of the macros. An example job description is given at the end of each specification.

PARAMETERS

Each macro is called by a statement of the form:

macroname ∇ *parameter 1, parameter 2, parameter 3,...*

Many of the parameters have the same meaning for each macro. These parameters, with their definitions, are listed below:

<i>ident</i>	is the 4-character identifier described above.
<i>pd</i>	is PD if the user wishes to supply his own program description. If this parameter is omitted, a default program description is incorporated automatically.
<i>label</i>	specifies the label in the job description to which control is passed in the event of any error. If this parameter is omitted, the job continues at the next command after the macro.
<i>in</i>	specifies the job description peripheral. It may be CR or TR, specifying a card or paper tape reader, respectively. If this parameter is omitted, CR is assumed.

time specifies the object program execution mill time limit (in seconds). The job will be terminated if this limit is exceeded. If this parameter is omitted, 50 is assumed.

disc specifies the type of backing store used for all files. This parameter may be ED (for E.D.S. backing store) or FD (for F.D.S. backing store); if it is omitted, ED is assumed.

comp is the compiler name; it can be XFAE or XFAT, as appropriate. If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE, if required, as suggested in the introduction to this Chapter.

If any other parameter is used, it is defined in the specification of the macro where it occurs.

The order of parameters is determined in the specification of the macro concerned. Parameters are each terminated by a comma and, if a parameter is omitted, its terminating comma must be present if there are any subsequent parameters present in the list. Trailing commas are unnecessary.

FORTTRAN

Function

The macro FORTTRAN is used to compile, consolidate and run a FORTRAN source program input on cards or paper tape.

Parameters

The macro is called by a statement of the form

FORTTRAN *ident, pdo, label, in, time, disc, comp*

where	<i>ident</i>	is the 4-character identifier
	<i>pdo</i>	is PD or OV depending on whether the user supplies his own program description for a non-overlaid or overlaid program respectively. If this parameter is omitted, the standard default program description will be incorporated automatically
	<i>label</i>	is error label in job description (optional)
	<i>in</i>	is CR or TR (optional)
	<i>time</i>	is the mill time limit in seconds (optional)
	<i>disc</i>	is ED or FD (optional)
	<i>comp</i>	is XFAE or XFAT (optional)

ident, label, in, time, disc and *comp* are described more fully in the section *Parameters*, page 74.

Standard default program description

```
PROGRAM (ident)
INPUT 1,5 = in0
OUTPUT 2,6 = LP7
TRACE 2
END
```

Note: It is possible to alter the level of tracing to TRACE 1 without supplying a program description, by including the statement TRACE 1 as an intersegment statement before the first segment. This facility is available only with the macro FORTTRAN; it results in some saving of core store, though even more core store will be saved if a program description is supplied including the statement TRACE 1 or TRACE 0.

Restrictions

- 1 Card or paper tape input must be punched in 1900 code
- 2 Mixed backing store not permitted
- 3 Source input must begin on the same peripheral as the job description
- 4 PAUSE statements must not appear in the source program
- 5 Trace steering lists must not be provided, nor is the facility for tracing every statement available

Documents required

FORT*ident*
DATA*ident*

Disc files required

PROGRAM ∇ XFAE,	containing the compiler #XFAE and consolidator #XPCK
SUBGROUPSRF7,	containing the SRF7 library group

FORTTRAN

ICLA-DEFAULT, to receive semicompiled output
PROGRAMV\FORT, a work file to receive binary output

Example

```
JOB CTL1, 1234, CHRIS
FORTTRAN CTLP , , , , , XFAE
****

DOC FORTCTLP
    TRACE 1
    MASTER DES
    Source
    FINISH
****

DOC DATACTLP
    Data
****
```

This job description will cause the source document, FORTCTLP, on cards, to be compiled and consolidated. The program will run using data from the document DATACTLP, also on cards, and results will be output on a line printer. If an error occurs or if the time taken to run the object program exceeds 50 seconds, the job will be terminated. The program will run with TRACE 1 instead of TRACE 2 as supplied by the default program description. The backing store to be used for all files is E.D.S.

FORTRANCOMP

Function

The macro FORTRANCOMP compiles a FORTRAN source program using #XFAE, consolidates it and stores the binary program on a disc file.

Parameters

The macro is called by a statement of the form

FORTRANCOMP ∇ *ident, pd, label, in, disc, comp*

where *ident* is the 4-character identifier
 pd is PD (optional)
 label is error label in job description (optional)
 in is CR or TR (optional)
 disc is ED or FD (optional)
 comp is XFAE or XFAT (optional)

These parameters are described more fully in the section *Parameters*, page 74.

Standard default program description

PROGRAM (*ident*)
INPUT 1,5 = *in*0
OUTPUT 2,6 = LP7
END

The level of tracing incorporated is the default level for #XFAE, that is, TRACE 1.

Restrictions

- 1 Card or paper tape input must be punched in 1900 code
- 2 Mixed backing store not permitted
- 3 Source input must begin on the same peripheral as the job description

Documents required

FORT*ident*

Disc files required

PROGRAM ∇ XFAE,	containing the compiler #XFAE and consolidator #XPCK
SUBGROUPSRF7,	containing the SRF7 library group
ICLA-DEFAULT,	to receive semicompiled output
PROGRAM ∇ <i>ident</i> ,	a work file to receive the binary program

Example

JOB CTL2, 1234, CHRIS
FORTRANCOMP NORA,PD,1A, , , XFAE
IN ED (PROGRAM NORA)
LOAD #PRG1
IN /0 (DATA)
ENTER 7

FORTRANCOMP

AT OTHER DELETED,ENDJOB

DISPLAY 'PRG1 FAILED'

1A ENDJOB

DOC FORTNORA

LIST

PROGRAM (PRG1)

INPUT 1 = CR0

OUTPUT 2 = LP0

TRACE 2

END

MASTER FRED

Source for #PRG1

FINISH

DOC DATANORA

Data for #PRG1

Here the FORTRANCOMP macro has been employed to enable the user to write his own job description prior to running the object program. The job description in the example will cause the source program to be compiled and consolidated, and the object program will be entered at entry point 7.

Entering the object program at this point causes a trace of every statement; hence a program description containing a TRACE 2 statement has been supplied. The program description has also been used to give the program a name, PRG1, which differs from the file name PROGRAM NORA. If the macro fails, control is passed to label 1A and the job is ended without running the object program.

FORTRANCOMPX

Function

The macro FORTRANCOMPX is used to compile a source program held on backing store, for example, one that has been developed under the Batch Development System (see page 87). If it is also required to run the object program, the macro FORTRANRUN (see page 83) should be incorporated after FORTRANCOMPX.

Parameters

The macro is called by a statement of the form

FORTRANCOMPX *bs, sourcefile, ident, pd, label, disc, comp*

where	<i>bs</i>	may be ED, FD or MT, and specifies the type of backing store holding the source file
	<i>sourcefile</i>	is the name of the source file. It takes the same form as in the READ FROM statement, that is, $f(g)$ or $f(g).s(g_s)$ if the file is on disc, and $f(g).s(g_s)$ if the file is on magnetic tape (see <i>Compiling from disc</i> and <i>Compiling from magnetic tape</i> , pages 26 and 27 respectively)
	<i>ident</i>	is the 4-character identifier
	<i>pd</i>	is PD (optional)
	<i>label</i>	is error label in job description (optional)
	<i>disc</i>	is ED or FD (optional)
	<i>comp</i>	is XFAE or XFAT (optional)

ident, pd, label, disc and *comp* are described more fully in the section *Parameters*, page 74.

Standard default program description

```
PROGRAM (ident)  
INPUT 1,5 = CR0  
OUTPUT 2,6 = LP7  
END
```

The level of tracing incorporated is the default level for #XFAE, that is, TRACE 1.

Restrictions

- 1 Card or paper tape input must be punched in 1900 code
- 2 Mixed backing store not permitted

Documents required

None

Disc files required

PROGRAM ∇ XFAE,	containing the compiler #XFAE and consolidator #XPCK
SUBGROUPSRF7,	containing the SRF7 library group
ICLA-DEFAULT,	to receive the semicompiled output
PROGRAM ∇ <i>ident</i>	a work file to receive the binary program

Example

```
JOB CTL6, 1234, CHRIS  
FORTRANCOMPX ED,BATCHFILEONE(6).PROG1,FRED , , , XFAE  
****
```

FORTRANCOMPX

This job description will cause the program FRED, held in the subfile PROG1 of the file BATCHFILEONE (generation number 6), to be compiled by #XFAE. The program will be consolidated and saved in binary form in the file PROGRAMVFRED.

FORTRANRUN

Function

The macro FORTRANRUN is used to run a binary program held on a disc file.

Parameters

The macro is called by a statement of the form

FORTRANRUN ∇ *ident, file, label, time, disc*

where	<i>ident</i>	is the 4-character identifier
	<i>file</i>	indicates the file (named PROGRAM ∇ <i>file</i>) where the program is to be found. If this parameter is omitted, a default name of PROGRAM ∇ FORT is assumed
	<i>label</i>	is error label in job description (optional)
	<i>time</i>	is the mill time limit in seconds (optional)
	<i>disc</i>	is ED or FD (optional)

ident, label, time and *disc* are described more fully in the section *Parameters*, page 74.

Standard default program description

Not applicable.

Restrictions

- 1 Card or paper tape input must be punched in 1900 code
- 2 Mixed backing store not permitted
- 3 PAUSE statements must not have appeared in the source program
- 4 Trace steering lists must not be provided, nor is the facility for tracing every statement available

Documents required

DATA*ident*

Disc files required

PROGRAM ∇ *file*, containing the binary program

Example

```
JOB CTL4, 1234, JOHN
FORTRANRUN MYRA,MYRA, ,20
****
DOC DATAMYRA
      Data
****
```

The program MYRA, held on the file PROGRAM MYRA, will be loaded and run. The limit on the execution mill time is 20 seconds.

FORTTRANSEMI

Function

The macro FORTTRANSEMI is used to compile a program, or segments of a program, and save the semicompiled output in a disc file.

Parameters

The macro is called by a statement of the form

FORTTRANSEMI ∇ *ident, scfile, pdo, label, in, disc, comp*

where	<i>ident</i>	is the 4-character identifier
	<i>scfile</i>	is the name of the file to which the semicompiled output is to be sent. It takes the same form as in the SEND TO statement, that is, $f(g)$ or $f(g).s(g_s)$. If <i>s</i> is not specified, the compiler chooses the default filename FORTSEMICOMP (see <i>The SEND TO statement</i> , page 21)
	<i>pdo</i>	is PD if the programmer wishes to supply his own program description. It is OV if the user wishes a default program description for overlay segments to be incorporated automatically. If this parameter is omitted, the default program description will specify non-overlay segments
	<i>label</i>	is error label in job description (optional)
	<i>in</i>	is CR or TR (optional)
	<i>disc</i>	is ED or FD (optional)
	<i>comp</i>	is XFAE or XFAT (optional)

ident, label, in, disc and *comp* are described more fully in the section *Parameters*, page 74.

Standard default program description

If parameter *pdo* is OV, the default program description is

OVERLAY SEGMENTS
END

If parameter *pdo* is omitted, the default program description is

SEGMENTS
END

The level of tracing incorporated is the default level for #XFAE, that is, TRACE 1.

Restrictions

- 1 Card or paper tape input must be punched in 1900 code
- 2 Mixed backing store not permitted
- 3 Source input must begin on the same peripheral as the job description

Documents required

FORT*ident*

Disc files required

PROGRAM ∇ XFAE,	containing the compiler #XFAE and consolidator #XPCK
<i>scfile</i> ,	to receive semicompiled output

FORTRANSEMI

Example

```
JOB CTE5,9234,CHRIS
FORTRANSEMI SUB1, SCLIB(2).SUB1, , , ,XFAE
****
DOC FORTSUB1
    SUBROUTINE SUB1
        Source
    FINISH
****
```

The subroutine SUB1 will be compiled, and a subfile named SUB1 will be created in the file SCLIB (generation number 2) to hold the semicompiled output.

BATCH DEVELOPMENT SYSTEM

General description

The FORTRAN Batch Development System provides a method of updating, compiling, and running a batch of programs that are being developed concurrently, either by one programmer or by a group of programmers.

Source programs and data files may be created and updated on E.D.S., F.D.S. or magnetic tape. In the case of disc files, the System employs the disc editor #XMED (see Chapter 10, page 51) to insert source programs and data into individual subfiles, and to amend the files using amendments submitted on cards or paper tape. In the case of magnetic tape files, the System employs the magnetic tape editor #XMUM. A specification of #XMUM may be found in the manual *Compiling Systems*; however, #XMUM is very similar to #XMED and in general the user of the Batch Development System need be familiar only with the remarks made in this section.

The two Batch Development System macros used with magnetic tape files also cause a *selective disc file* to be created. This disc file will contain the most recent versions of all the programs on the magnetic tape file which have been referenced during the current run (that is, by insertion or amendment) unless their transfer to the disc file has been explicitly suppressed (see below). The programs on the selective disc file may then be compiled by #XFAE and run exactly as if the Batch Development System file had been held on disc, thus combining the storage advantage of magnetic tape with the greater efficiency of disc usage.

The macros used in the System are as follows:

<i>Macro name</i>	<i>Function</i>
NEWBATCH	Creates a batch of source programs and data on disc.
NEWBATCHMT	Creates a batch of source programs and data on magnetic tape.
AMENDBATCH	Updates a batch of source programs and data on disc.
AMENDBATCHMT	Updates a batch of source programs and data on magnetic tape.
FORTBATCH	Compiles, consolidates and runs a batch on E.D.S.
FORTBATCHFD	Compiles, consolidates and runs a batch on F.D.S.

A file of source programs is created using the macro NEWBATCH or NEWBATCHMT. Periodically, amendments to programs in the file may be batched together on paper tape or cards, and the macro AMENDBATCH or AMENDBATCHMT, as appropriate, is used to produce an updated file. The macro FORTBATCH or FORTBATCHFD may then be applied to the disc file produced by the latest amendment, in order to process the programs in succession. Each program will be compiled, consolidated and run immediately if there are no compilation errors. It is not necessary to write individual job descriptions for each program.

The source file may be updated, and the amended programs then run, any number of times. New programs may also be inserted at any time. When a program has been fully tested, a binary dump may be obtained by a separate run using the standard macro FORTRANCOMPX (see page 81).

Operation of the System

The Batch Development System is composed of three programs: the editor program, #XMED (or #XMUM for magnetic tape); the FORTRAN 16K disc compiler #XFAE; and the consolidator #XPCK. If desired, both #XMED and #XMUM may be held on the same disc file together with the compiler and consolidator. This is an efficient measure if a selection of Batch Development System macros is being used.

The running of the System consists of two phases: the *edit phase* and the *process phase*.

In the edit phase the editor is used either to create a new source file or to amend the current file of programs and data. For any job in which both source and data are to be edited, the parameters for editing the source must precede those for editing the data.

All the facilities of #XMED are available, except that the directives *CPCOPY, *TPCOPY and *SWITCH must not be used (see Chapter 10, page 51).

If the editor #XMUM is used, all directives may be employed except *ORDER, *OFFLINE, *CPCOPY, *TPCOPY, *SWITCH, and *RESTART.

In the process phase each program in the new or updated file that was referenced during the edit phase is processed as follows:

- 1 The 16K disc compiler is loaded into core and the program compiled

- 2 If the compilation is error free, the compiler calls in the consolidator which consolidates the program and loads it into core
- 3 The object program is entered

Each of these events takes place under the control of GEORGE.

The System will edit, compile and run only those programs which are referenced in *FORTRAN lines in the current directives to the editor (see *Macro specifications*, below).

If a program is to be edited, but it is wished to suppress compilation (and hence, if #XMUM is being used, copying to the selective file) the *NOCOMPILATION directive (format: *NOCOMPILATION) may be used. This directive must immediately follow the *FORTRAN directive.

Programming information

The facilities available to the programmer when compiling a single program are also available under the Batch Development System, although sometimes in a different manner. If the programmer does not supply a program description with a program in the batch, a standard default program description (see below) is incorporated automatically. If facilities other than those provided by the default option are required, the programmer must supply his own program description.

TRACING

Special forms of the TRACE 2 statement are allowed in program descriptions for the Batch Development System.

If a trace steering list is to be used to control the output of trace information, the statement

TRACE 2S

should be specified in the program description. The steering list itself should be inserted at the beginning of the data subfile for the program.

If all the statements are to be traced, the TRACE statement should be written

TRACE 2A

and no trace steering list is required.

DATA HANDLING

Programmers may use any number of files on disc or magnetic tape for input or output of data, or as work files. Basic peripherals other than the line printer should not be used, but a set of data for use by a program may be inserted in a data subfile in the batch file, immediately after the source program. This subfile, known as the *system input file*, is used for data that would normally be input via a card or paper tape reader. To access the system input file a special form of program description statement is available.

INPUT k = SYSTEM

where k is a channel number or list of channel numbers referred to in the source program, as described in the section *Channel description statements*, page 9. This statement will cause any READ statement specifying k to reference the data held in the data subfile for that program. Execution error 23 will be reported if any of the statements WRITE, REWIND, BACKSPACE, or ENDFILE specify the system input channel; execution error 06 will be reported if an attempt is made to read beyond the end of data on this channel.

The normal output channel is the line printer, which is specified in the usual way. Alternatively, data output to the line printer may be off-lined via the GEORGE output file. (This facility is available only with GEORGE 2.)

Other input or output files on disc or magnetic tape are specified in the normal way. The system input channel has the unit number 15, that is, ED15 or FD15. Care should therefore be taken not to specify this unit number for any other file on the same medium.

Programs will terminate either when a fatal error is detected or when a STOP statement is encountered.

Macro specifications

There follow sections common to all the Batch Development System macros, succeeded by specifications of the macros in alphabetical order.

STANDARD DEFAULT PROGRAM DESCRIPTION

When using the Batch Development System the program description may be omitted, in which case the following program description is incorporated automatically:

```
PROGRAM (name)  
INPUT 1,5 = SYSTEM  
OUTPUT 2,6 = LPO  
TRACE 2  
END
```

where *name* is assumed to be the first twelve characters of the job subfile name specified in the *FORTRAN directive.

If the default program description is not sufficient the user must provide his own.

RESTRICTIONS

There are few restrictions on the programs that may be developed under this System: they may be overlaid or non-overlaid, extended or compact. All the program description facilities described in Chapters 1, 3, 5, 6, 8 and 9 are available and object programs may use any number of disc or tape files for input or output of data, or as workspace.

The initial statements SEND TO (page 21), DUMP ON (page 23) and RUN (page 23) must not be present. The READ FROM statement (page 25) is not allowed since the program must be held entirely in the source input file. All other initial and intersegment statements have their usual effect.

The documents submitted to the System must be input on the same peripheral as the job description.

If a program description is supplied, it must not reference any basic peripheral other than the line printer, or any disc file with unit number 15 (see above).

PAUSE statements should not be used to terminate programs run under the Batch Development System. They have the same effect as CONTINUE: that is, control is passed to the next statement.

Within any one batch of programs each subfile name must be unique.

PARAMETERS

Each macro is called by a statement of the form:

macroname ∇ *parameter 1, parameter 2, parameter 3, . . .*

The order of parameters is determined by the specification of the macro concerned. Parameters are each terminated by a comma and, if a parameter is omitted, its terminating comma must be present if there are any subsequent parameters present in the list. Trailing commas are unnecessary.

AMENDBATCH

Function

The macro AMENDBATCH causes an existing batch of programs on a disc file to be updated and written to a new disc file. Source programs or data or both may be edited, and new source programs and data may be inserted.

Parameters

The macro is called by a statement of the form

AMENDBATCH *peri, docname, oldfile, newfile, medium, comp*

- where
- | | |
|----------------|--|
| <i>peri</i> | is TR or CR according to the peripheral (paper tape reader or card reader) from which the amendments and insertions are to be input |
| <i>docname</i> | is the name of the document containing the amendments and insertions |
| <i>oldfile</i> | is the name of the disc file containing the existing batch. It may be followed by the file generation number in parentheses |
| <i>newfile</i> | is the name of the disc file which will contain the amended batch. It may take any of the following forms:
$f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings (see, e.g., *OUT, page 53) |
| <i>medium</i> | is ED or FD according to the medium (E.D.S. or F.D.S.) of <i>newfile</i> and the file containing the editor #XMED. If this parameter is omitted, ED is assumed |
| <i>comp</i> | is the compiler name (XFAE or XFAT). If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE if required (see the introduction to this Chapter) |

Standard default program description

See page 89

Restrictions

See page 89

Documents required

Input must consist of a document of the form

DOCUMENT *docname*, Txxx

Amendments

////

Blank line

- where
- | | |
|-------------------|---|
| <i>docname</i> | is as defined in the macro call |
| <i>Amendments</i> | will be references to FORTRAN source programs and, optionally, data |
| Txxx | defines the four characters xxxx which are to serve as a replacement for **** as the document terminator recognized by GEORGE |

A reference to a FORTRAN source program must begin with the directive

*FORTRAN *s, u*

- where
- | | |
|----------|--|
| <i>s</i> | is the subfile name, consisting of up to eight alphanumeric characters, hyphens or spaces, and beginning with a letter. <i>s</i> must be unique within any one batch of programs |
| <i>u</i> | is any desired user identification |

AMENDBATCH

Following the *FORTRAN directive, a complete source program may be inserted, or a source program already in the file may be updated using the *ALTER directive (see page 55). The reference to the source program must be concluded by the subfile terminator **** (see page 53).

Any reference to data connected with a source program must be placed immediately after the reference to that program and must begin with the directive

*DATA ∇ s' , u'

where s' is the data subfile name and takes the form DATA s (where s is as defined above)

u' is any desired user identification

A set of data may be inserted, or a pre-existing set may be amended using the *ALTER directive, as described above. The reference to data must be concluded by the subfile terminator ****.

All alterations to programs or data must be in order of line number, and alterations to the source program must precede alterations to the data.

Disc files required

PROGRAM ∇ XFAE, containing the compiler #XFAE, consolidator #XPCK, and editor #XMED

Example

See *Example 3*, page 102.

AMENDBATCHMT

Function

The macro AMENDBATCHMT causes an existing batch of programs in a magnetic tape file to be updated and written to a new magnetic tape file. Source programs or data or both may be edited, and new source programs and data may be inserted. Every program or block of data referred to in the run will be written to a selective disc file, which may then be processed using the macro FORTBATCH or FORTBATCHFD.

Parameters

The macro is called by a statement of the form

AMENDBATCHMT *peri, docname, oldMTfile, newMTfile, selectfile, medium, comp*

where	<i>peri</i>	is TR or CR according to the peripheral (paper tape reader or card reader) from which the amendments and insertions are to be input
	<i>docname</i>	is the name of the document containing the amendments and insertions
	<i>oldMTfile</i>	is the name of the magnetic tape file containing the existing batch. It may be followed by the file generation number in parentheses
	<i>newMTfile</i>	is the name of the magnetic tape file which will contain the amended batch. It may take any of the following forms: $f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings (see, e.g., *OUT, page 53)
	<i>selectfile</i>	is the name of the selective disc file which will contain the amended batch. It may take any of the following forms: $f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings This parameter may be omitted, in which case the default name COMPILEFILE is assumed
	<i>medium</i>	is ED or FD according to the medium (E.D.S. or F.D.S.) of the selective disc file, the file containing the editor #XMUM, and the #XMUM workfile. If this parameter is omitted, ED is assumed
	<i>comp</i>	is the compiler name (XFAE or XFAT). If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE if required (see the introduction to this Chapter)

Standard default program description

See page 89

Restrictions

See page 89

Documents required

Input must consist of a document of the form

DOCUMENT *docname*, Txxxx

Amendments

////

Blank line

AMENDBATCHMT

where *docname* is as defined in the macro call
Amendments will be references to FORTRAN source programs and, optionally, data
Txxxx defines the four characters *xxxx* which are to serve as a replacement for **** as the document terminator recognized by GEORGE

A reference to a FORTRAN source program must begin with the directive

FORTRANs, u*

where *s* is the subfile name, consisting of up to eight alphanumeric characters, hyphens or spaces, and beginning with a letter. It must be unique within any one batch of programs
u is any desired user identification

Following the *FORTRAN directive, a complete source program may be inserted, or a source program already in the file may be updated using the *ALTER directive (which takes the same form as that used with #XMED, see page 55). The reference to the source program must be concluded by the subfile terminator **** (see page 53).

Any reference to data connected with a source program must be placed immediately after that program and must begin with the directive

DATAs', u'*

where *s'* is the data subfile name and takes the form *DATAs* (where *s* is as defined above)
u' is any desired user identification

A set of data may be inserted, or a pre-existing set may be amended using the *ALTER directive, as described above. The reference to data must be concluded by the subfile terminator ****.

All alterations to programs or data must be in order of line number, and alterations to the source program must precede alterations to the data.

All FORTRAN source programs and data mentioned in the document will normally be copied to the selective disc file, *selectfile*, for subsequent compilation and running using the macro FORTBATCH or FORTBATCHFD. If it is required to update or insert a program without copying it to the selective disc file, the *NOCOMPILATION directive (format: *NOCOMPILATION) must be input immediately after the *FORTRAN directive.

Disc files required

PROGRAM *∇* XFAE, containing the compiler #XFAE, consolidator #XPCCK, and editor #XMUM

Example

See *Example 4*, page 103.

FORTBATCH FORTBATCHFD

Function

The macros FORTBATCH and FORTBATCHFD control the batch compilation and running of programs present in the disc files produced by other Batch Development System macros. FORTBATCH is for use with E.D.S. files and FORTBATCHFD is for use with F.D.S. files.

If the file concerned was produced by the macro NEWBATCH or NEWBATCHMT, all programs present will be compiled; if the macro AMENDBATCH or AMENDBATCHMT was used, only those programs referenced in the amendments document will be compiled. In the case of AMENDBATCHMT, compilation can be suppressed using the *NOCOMPILATION directive (see page 94).

Parameters

The macros are called by a statement of the form:

FORTBATCH or }
FORTBATCHFD } ∇ *inputfile, workfile1, workfile2, comp*

where	<i>inputfile</i>	is the name of the disc file containing the latest version of the batch. It may be followed by the file generation number in parentheses. If the batch was produced by the macro NEWBATCH or AMENDBATCH, then normally <i>inputfile</i> $\equiv$ <i>newfile</i> ; if NEWBATCHMT or AMENDBATCHMT was used, then <i>inputfile</i> $\equiv$ <i>selectfile</i>
	<i>workfile1</i>	is the name of the first disc file to be used as a work file by the compiler and consolidator. If this parameter is omitted, the default file name ICLA-DEFAULT will be assumed
	<i>workfile2</i>	is the 12-character name of the second disc work file for the compiler and consolidator. This work file is necessary only if there are overlay programs present in the batch. The first eight characters of the name must be PROGRAM ∇ . If this parameter is omitted, the default file name PROGRAM ∇ FORT will be assumed
	<i>comp</i>	is the compiler name (XFAE or XFAT). If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE if required (see the introduction to this Chapter)

Note: If *workfile 1* is omitted, then *workfile 2* must be omitted also.

Standard default program description

See page 89

Restrictions

See page 89

Documents required

Not applicable

Disc files required

PROGRAM ∇ XFAE,	containing the compiler #XFAE and consolidator #XPCK
SUBGROUPSRF7,	containing the SRF7 library group
ICLA-DEFAULT,	to receive semicompiled output
PROGRAM ∇ FORT,	a work file to receive binary output

Example

See *Examples 3 and 4*, page 102

NEWBATCH

Function

The macro NEWBATCH is used to create a file of source programs and data on disc.

Parameters

The macro is called by a statement of the form

NEWBATCH ∇ *peri, docname, newfile, medium, comp*

where	<i>peri</i>	is TR or CR according to the peripheral (paper tape reader or card reader) from which the programs and data are to be input
	<i>docname</i>	is the name of the document containing the programs and data
	<i>newfile</i>	is the name of the disc file which will contain the newly created batch. It may take any of the following forms: $f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings (see, e.g., *OUT, page 53)
	<i>medium</i>	is ED or FD according to the medium (E.D.S. or F.D.S.) of <i>newfile</i> and the file containing the editor #XMED. If this parameter is omitted, ED is assumed
	<i>comp</i>	is the compiler name (XFAE or XFAT). If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE if required (see the introduction to this Chapter).

Standard default program description

See page 89

Restrictions

See page 89

Documents required

Input must consist of a document of the form

DOCUMENT *docname*, Txxxx

Insertions

////

Blank line

where	<i>docname</i>	is as defined in the macro call
	<i>Insertions</i>	will be references to FORTRAN source programs and, optionally, data
	Txxxx	defines the four characters xxxx which are to serve as a replacement for **** as the document terminator recognized by GEORGE.

A reference to a FORTRAN source program takes the form

*FORTRAN ∇ *s, u*

Source program

where	<i>s</i>	is the subfile name, consisting of up to eight alphanumeric characters, hyphens or spaces, and beginning with a letter. It must be unique within any one batch of programs
	<i>u</i>	is any desired user identification
	****	is the subfile terminator (see page 53)

NEWBATCH

Any reference to data connected with a source program must be placed immediately after that program, and takes the form

`*DATA $\nabla$ s', u'`

Data

`****`

where s' is the data subfile name and takes the form DATA s (where s is as defined above)

u' is any desired user identification

`****` is the subfile terminator

The source program will be inserted in the subfile s , and the data will be inserted in the subfile s' . These subfiles may subsequently be updated by means of the macro AMENDBATCH (see page 91).

Disc files required

PROGRAM ∇ XFAE, containing the compiler #XFAE, consolidator #XPCK and editor #XMED

Example

See *Example 1*, page 101

NEWBATCHMT

Function

The macro NEWBATCHMT is used to create a file of source programs and data on magnetic tape.

Parameters

The macro is called by a statement of the form

NEWBATCHMT *peri, docname, newMTfile, selectfile, medium, comp*

where	<i>peri</i>	is TR or CR according to the peripheral (paper tape reader or card reader) from which the programs are to be input
	<i>docname</i>	is the name of the document containing the programs
	<i>newMTfile</i>	is the name of the magnetic tape file which will contain the newly created batch. It may take any of the following forms: $f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings (see, e.g., *OUT, page 53)
	<i>selectfile</i>	is the name of the selective disc file which will contain the newly created batch. It may take any of the following forms: $f, f(g), f(g) = (g'), f(g) = f'(g')$ where the symbols have the usual meanings. This parameter may be omitted, in which case the default name COMPILEFILE is assumed
	<i>medium</i>	is ED or FD according to the medium (E.D.S. or F.D.S.) of the selective disc file, the file containing the editor #XMUM, and the #XMUM workfile. If this parameter is omitted, ED is assumed
	<i>comp</i>	is the compiler name (XFAE or XFAT). If this parameter is omitted, it is assumed by default to be XFAT. The default value can be changed to XFAE if required (see the introduction to this Chapter)

Standard default program description

See page 89

Restrictions

See page 89

Documents required

Input must consist of a document of the form

DOCUMENT *docname*, Txxxx

Insertions

////

Blank line

where	<i>docname</i>	is as defined in the macro call
	<i>Insertions</i>	will be references to FORTRAN source programs and, optionally, data
	Txxxx	defines the four characters xxxx which are to serve as a replacement for **** as the document terminator recognized by GEORGE

NEWBATCHMT

A reference to a FORTRAN source program takes the form

```
*FORTRAN $\nabla$ s, u
      Source program
****
```

where s is the subfile name, consisting of up to eight alphanumeric characters, hyphens or spaces, and beginning with a letter. It must be unique within any one batch of programs

u is any desired user identification

**** is the subfile terminator (see page 53).

Any reference to data connected with a source program must be placed immediately after that program, and takes the form

```
*DATA $\nabla$ s', u'
      Data
****
```

where s' is the data subfile name and takes the form DATAs (where s is as defined above)

u' is any desired user identification

**** is the subfile terminator

The source program will be inserted in the subfile s , and the data will be inserted in the subfile s' . These subfiles may subsequently be updated by means of the macro AMENDBATCHMT (see page 93).

All FORTRAN source programs and data mentioned in the document will normally be copied to the selective disc file, *selectfile*, for subsequent compilation and running using the macro FORTBATCH or FORTBATCHFD. If it is required to create a program without copying it to the selective disc file, the *NOCOMPILATION directive (format: *NOCOMPILATION) must be input immediately after the *FORTRAN directive.

Disc files required

PROGRAM ∇ XFAE, containing the compiler #XFAE, consolidator #XPCK and editor #XMUM

Example

See Example 2, page 101

EXAMPLES OF USE OF THE BATCH DEVELOPMENT SYSTEM

Example 1

```
JOB XYZ1,37124,CML
NEWBATCHVCR,CLDOC,BATCHFILE5,,XFAE
****
DOCUMENT CLDOC,T???
*FORTRANVPROG1,J.A.SMITH
    MASTER MAINSEG
    Source
    FINISH
****
*DATAV DATAPROG1,J.A.SMITH
1 2 3 4
1000.312
1000.379
1000.917
1000.999
****
*FORTRANVPROG2,JANE
    MASTER JANE
    Source
    FINISH
****
////
Blank line
???
```

This job will copy the two programs PROG1 and PROG2 from the card document CLDOC to the disc file BATCHFILE5, together with the data for PROG1. //// followed by a blank line is the terminator for the editing program #XMED. ??? has been specified as a document terminator alternative to the standard ****, otherwise GEORGE would have recognized the subfile terminator as a document terminator.

The file that has now been created may be edited, using the macro AMENDBATCH or compiled using the macro FORTBATCH.

Example 2

```
JOB XYZ2,37124,CML
NEWBATCHMTVCR,CLDOC,BATCHFILE6,DISCFIL,,XFAE
****
DOCUMENT CLDOC,TZZZZ
*FORTRANVPROG1,J.A.SMITH
    MASTER MAINSEG
    Source
    FINISH
****
```

*DATA∇DATAPROG1,J.A.SMITH

1 2 3 4

1000.312

1000.379

1000.917

1000.999

*FORTRAN∇PROG2,JANE

MASTER JANE

Source

FINISH

////

Blank line

ZZZZ

This job is similar to *Example 1* but illustrates the differences in document structure when a batch is created on magnetic tape. The two programs PROG1 and PROG2, together with the data for PROG1, will be copied from the card document CLDOC to the magnetic tape file BATCHFILE6. The programs and data will also be copied to the selective E.D.S. file DISCFIL which may then be compiled using the macro FORTBATCH. The file BATCHFILE6 may be updated using the macro AMENDBATCHMT.

Example 3

JOB UPDATE,321,JACK JONES

AMENDBATCH∇CR,AMEN1,BATCHFILE5,BATCH6,,XFAE

FORTBATCH∇BATCH6,,XFAE

DOCUMENT AMEN1, TAAAA

*FORTRAN∇PROG1, J.A.S.

*ALTER∇2,1

DIMENSION A(10,5)

*FORTRAN∇PROG2,JANE

*FORTRAN∇PROG3,JILL

PROGRAM(J123)

INPUT 2 = SYSTEM

OUTPUT 3 = LP7

TRACE 0

END

MASTER TEST 1

Source

FINISH

```

*DATA∇DATAPROG3,JILL
1 2 3 4 0.1 0.2 0.3 0.4
****
*FORTRAN∇PROG7,J.SMITH,ROOM 17
****
*DATA∇DATAPROG7,J.SMITH,ROOM 17
*ALTER∇1,1
      6      12
****
////
Blank line
AAAA

```

The batch of programs held in the E.D.S. file BATCHFILE5 is to be updated. Amendments and insertions appear in the card document AMEN1. The first program (PROG1) in BATCHFILE5 will be amended, and copied to the new file BATCH6. PROG2 will be copied unchanged to the new file, but will be compiled when the batch is processed. A new program PROG3, together with its data will be added to the batch in the new file. The data for PROG7 will be changed and the program recompiled and run. (The dummy reference to the source of PROG7 is necessary if the program is to be compiled and run.)

Programs not mentioned in the document are copied across but not compiled.

The programs PROG1, PROG2, PROG3 and PROG7 will be compiled by the 16K disc compiler and the object programs will be run. Double buffering (see page 10) has been specified for the line printer, and trace level 0 (see page 39) has been adopted to save some core store.

Example 4

```

JOB UPDATE,322,JACK JONES
AMENDBATCHMT∇CR,AMEN1,FORTRANJOBS(3),FORTRANJOBS(2)=(4),FORTPROGS,FD,XFAE
FORTBATCHFD∇FORTPROGS,,,XFAE
****
DOCUMENT AMEN1,TAAAA
*FORTRAN∇PROG1,J.A.S.
*ALTER∇2,1
      DIMENSION A(10,5)
****
*FORTRAN∇PROG2, JANE
****
*FORTRAN∇PROG3, JILL
*NOCOMPILATION
      PROGRAM (J123)
      INPUT 2 = SYSTEM
      OUTPUT 3 = LP0
      END
      MASTER TEST 1
Source
      FINISH

```

```

****
*DATA∇DATAPROG3,JILL
1 2 3 4 0.1 0.2 0.3 0.4
****
*FORTRAN∇PROG7, J.SMITH, ROOM 17
****
*DATA∇DATAPROG7, J. SMITH, ROOM 17
*ALTER∇1, 1
        6      12
****
////
Blank line
AAAA

```

In this example a batch of programs is held on the magnetic tape file FORTRANJOBS (generation number 3). Amendments and insertions appear in the card document AMEN1. The program PROG1 will be amended and written to the file FORTRANJOBS which currently has generation number 2, and is to be given generation number 4. The program PROG2 will be copied to this file unchanged, but will be compiled when the batch is processed. A new program PROG3, together with its data, will be inserted in the new file. The program PROG7 will be copied across, together with an amended version of its data. (The dummy reference to the source of PROG7 is necessary if the program is to be compiled and run.)

Programs not mentioned in the document will be copied across but not compiled.

The programs PROG1, PROG2 and PROG7 will also be copied to the selective F.D.S. file FORTPROGS. PROG3 will not be copied because the *NOCOMPILATION directive was given. The programs on the file FORTPROGS will be compiled by the 16K disc compiler and the object programs will be run.

USER JOB DESCRIPTIONS

The user may require more complex facilities than can be provided with a standard macro. In this case, the user may write his own job description, without using standard ICL macros.

For example, the user might write his own job description in the following cases:

- 1 If there is more than one source document in the compiler input stream
- 2 If input to the compiler is not on the job description peripheral
- 3 If Atlas or EBCDIC card code is used
- 4 If source or data on magnetic tape is to be specified in the job description
- 5 If a trace steering list is supplied, or trace information is required for every statement in the object program
- 6 If special action is to be taken as a result of STOP or PAUSE statements in the object program
- 7 If special action is to be taken when the compiler or consolidator halts
- 8 If a suite of FORTRAN programs is to be run
- 9 If the compiler and consolidator are resident in files of different names

In order to write a job description that caters for all possible events, the user must know the operating characteristics of the compiler, consolidator, and object program, including the unit numbers of peripherals used; console messages output by each program; and switches that may be used in exception conditions. Descriptions of each of these can be found in Chapter 14, page 113. Information on program testing, including the use of the trace package, can be found in Chapter 7, page 33.

The READ FROM statement may be used to switch the medium of the input stream to the compiler, but care must be taken when using this facility under GEORGE. When #XFAE encounters a READ FROM statement, it releases the peripheral that was previously in use and allocates the new peripheral. Since GEORGE does not release its job description peripheral during the running of a batch, a peripheral declared as /0 will not be released. If the new peripheral is not specified in the GEORGE job description, it will be picked up by GEORGE on-line.

Examples

The following are examples of job descriptions that can control the #XFAE compiler and FORTRAN object programs.

- 1 A program input on paper tape is to be compiled and consolidated. The job description consists of the following statements on cards:

```
JOB FORTTEST,1234,K.BAKER
IN ED (PROGRAM DISC)
LOAD #XFAE
IN TRO (TAPE INPUT)
ENTER 0
1 AT DELETED FI,GO TO 10
AT HALTED ST,GO TO 11
AT FAIL,END
11 HALT 'NEEDS MORE STORE'
RESUME,1
10 IN ED (PROGRAM XPCK)
LOAD #XPCK
RESUME
2 AT DELETED HH,END
AT HALTED ST,GO TO 12
AT FAIL,END
```

12 HALT 'NEEDS MORE STORE'

RESUME,2

ENDJOB

The compiler XFAE is loaded from the file PROGRAM DISC and input to XFAE is from a document headed TAPE INPUT. The compiler is entered, and if the compilation is error free, the consolidator is loaded from the file PROGRAM XPCK, and the program consolidated. If either the compiler or consolidator needs more core, the message NEEDS MORE STORE is typed to the operator. If the compilation contains errors, or the consolidation fails, the job is terminated. This job requires a file into which the consolidator can dump the object program. A DUMP ON statement must therefore be input as an initial statement (see *The DUMP ON statement*, page 23).

- 2 A FORTRAN program is to be run, using input from two paper tape readers, and producing output on magnetic tape. If this program runs successfully, a second program will be run using the information on the magnetic tape as input, and outputting results to a line printer.

The program description of the first program is:

PROGRAM (CLTP)

INPUT 1 = TR0

INPUT 2 = TR1

OUTPUT 3 = MT0 (RESULTS FILE)

TRACE 2

END

The program description of the second program is:

PROGRAM (TCPL)

INPUT 1 = MT0 (RESULTS FILE)

OUTPUT 2 = LP0

TRACE 2

END

The job description required to run the job is:

JOB TEST 1,1423, I.MOORE

33 DATA MT0 (73201,RESULTS FILE)

34 DATA TR0 (INPUT 1)

35 DATA TR1 (INPUT 2)

IN ED (PROGRAM FILE)

LOAD #CLTP

IN 34,35

OUT 33

ENTER 0

AT HALTED 77,GO TO 11

AT FAIL, END

11 IN ED (PROGRAM FILE)

LOAD #TCPL

IN 33

ENTER 0

END

The program #CLTP is loaded from the E.D.S. file PROGRAM FILE. The paper tape documents INPUT 1 and INPUT 2 are read, and a file produced on a magnetic tape with file name RESULTS FILE and tape serial number 73201. If the program runs successfully, the statement PAUSE 77 is processed; this produces a HALTED message. GEORGE then loads the second program, #TCPL, and runs it using the file RESULTS FILE as input. If either program fails, trace information is automatically output. Thus an event list for the second program is not necessary.

- 3 A program that has been previously compiled is to be run using a trace steering list. The steering list precedes the program's data in the document DATA, which is to be read on cards. The job description is as follows:

JOB CTEST,77271,R.MARTIN

IN ED (PROGRAM X3Y1)

LOAD #X3Y1

IN CR0 (DATA)

ENTER 8

55 AT HALTED 26,GO TO 66

AT HALTED 46, END

AT FAIL,END

66 SWITCH OFF 0

RESUME

AT HALTED 36, GO TO 77

AT FAIL,END

77 SWITCH ON 0

RESUME,55

END

The program is loaded from disc and entered, and the trace steering list, followed by the data, is read. When the statement PAUSE 26 is encountered, switch 0 of the program is turned off; this suppresses trace output. Trace output is restarted when the statement PAUSE 36 is encountered. The program is terminated at the statement PAUSE 46.

PART 2 OPERATING CONSIDERATIONS

Chapter 13 System description

This chapter is designed primarily for those responsible for the running of an installation and describes the preliminary steps necessary before using the 1900 FORTRAN 16K disc compiling system, with particular reference to disc file organisation. It is assumed that the reader has a knowledge of the file allocator programs #XJEC and #XJFC for E.D.S. and F.D.S. respectively, and those hardware aspects of E.D.S. and F.D.S. which the use of those programs involve. Full specifications of #XJEC and #XJFC are given in the manual *Direct Access Utilities*.

GENERAL DESCRIPTION

Compilation and consolidation

The FORTRAN 16K disc compiling system consists of the compiler #XFAE and the consolidator #XPCK. The compiler accepts as input source and semi-compiled segments on paper tape, cards, E.D.S., F.D.S. and magnetic tape. Compiler output consists of semi-compiled segments on disc together with an optional source listing on a line printer. Consolidation by #XPCK is an automatic process following the successful compilation of a complete program. At the end of compilation, the compiler deletes itself from core store, causing the consolidator to be loaded from disc, and passes the necessary steering information to the consolidator. Input to the consolidator consists of the semi-compiled segments on disc produced during compilation and those library routines contained in the FORTRAN disc compiler library group (the SRF7 group) specifically required by the program. Other subfiles on disc consisting of semi-compiled segments produced by previous compilations and additional disc libraries can be input to the consolidator if so specified by the user in the compiling system statements of the FORTRAN source program. Output from the consolidator consists of binary object program either loaded into core store ready for running or dumped on to a disc file in a loadable form, the choice being determined by the user in the source program. An optional line printer listing giving the sizes and positions of the various sections of the object program can be produced by #XPCK. When the object program is left in core store, the consolidator #XPCK changes its name to that of the object program.

Consolidation follows compilation only for complete, error free programs. Programs with errors or a group of segments, as distinct from a complete program, will not be consolidated following compilation. The consolidator #XPCK can be used as a separate program, but this should rarely be necessary for FORTRAN users.

The compiler #XFAE is overlaid from disc and therefore the disc cartridge from which it is loaded must be on-line throughout the compilation. The consolidator must also be loaded from a disc file when consolidation automatically follows compilation.

Object program

Object programs may be non-overlaid or overlaid, as specified by the user in the compiling system statements for the source program. In the latter case the program must be dumped on to a disc file by the consolidator and this file will be retained by the object program for overlay purposes during the run.

FORTRAN object programs will cause data to be input from and output to whatever peripherals the user specifies. The FORTRAN system allows the use of disc files and magnetic tape files for input and output in addition to basic peripherals. More than one peripheral of a given type may be used if required.

DISC FILE STRUCTURE

The 1900 FORTRAN 16K disc compiling system requires the use of a certain number of permanent files on disc; for example, files are required for holding the compiler, semi-compiled output and the FORTRAN library. Some of these files must be given standard names.

SYSTEM INITIALIZATION

A DISPLAY message giving the program name and an indication of the mill time used during compilation will be output on the console typewriter if word 30 of #XFAE has been given the value #10000000. The word may be

altered to this value either permanently, using the library routine #XPEU (details of which may be found in the manual *Compiling Systems*) or temporarily, for the compilation concerned, by the operator inputting the instruction ON #XFAE 2. This instruction will set bit 2 of word 30 to 1.

The format of the DISPLAY message is:

0#XFAE; DISPLAY:- COMPILED *m name a*

where *m* is a measure of the time used
name is the four-character program name
a is EC if there are no errors, and ZZ if any errors are found.

File allocation

All permanent files must be allocated space on a particular disc unit by use of one of the file allocator programs #XJEC or #XJFC for E.D.S. or F.D.S. files respectively. These programs allow the user considerable flexibility in defining the file, but it is recommended that files to be used in the FORTRAN system observe the following rules:

- 1 All files, apart from those only to be used for object program data, must be set up with a bucket size of 128 words. Object program data files may have any defined bucket size. In that case the larger the bucket size, the more efficient disc transfers will be, assuming that there is sufficient core store to allow the object program buffer connected with the file to be the same size as the bucket.
- 2 E.D.S. files should be allocated in units of 80 hardware blocks, i.e. one cylinder. Any files extended by the FORTRAN system (on either E.D.S. or F.D.S.) will be extended by a multiple of 80 blocks, wherever possible.
- 3 E.D.S. files should be allocated on adjacent cylinders and should be allocated on one disc cartridge only. Permanent files (on either E.D.S. or F.D.S.) will be extended only on the disc containing the current end of file.

Obligatory disc files

This section describes the files required by the compiler and consolidator. A suitable system for finding programs from E.D.S. or F.D.S. must be created on a disc unit; details of such a system are given in the specifications of #XPEB (for E.D.S.) or #XPFB (for F.D.S.) in the manual *Compiling Systems*. For the files required when running #XFAE under GEORGE, see Chapter 12, page 73.

COMPILER FILE

The compiler #XFAE and consolidator #XPCK should be inserted into a previously allocated compiler file using standard library manipulation programs (e.g. #XPEU). The compiler file may contain other programs. It must have a name of the form PROGRAMVyyyy where yyyy can be any four characters starting with a letter.

The compiler is then loaded into core by a message of the form:

FI #XFAE #yyyy

If yyyy is XFAE, this message can be reduced to:

FI #XFAE

At the end of a successful compilation of a complete program the compiler will delete itself and issue the message:

0#XFAE; DELETED:- FI#XPCK #yyyy

The compiler and consolidator together occupy about 200 buckets.

LIBRARY FILE

A file called SUBGROUPSRF7, of at least 720 blocks, must be allocated, and the FORTRAN 16K disc compiler library, the SRF7 group, inserted in it using #XPES. (A specification of #XPES is given in the manual *Compiling Systems*.)

The library file must be on the same medium as the compiler file.

Consolidation time is improved considerably if the library file is in directory format; the number of disc accesses during consolidation is then reduced. Directory format is obtained by means of the utility program #XFYZ, details of which may be found in the manual *Compiling Systems*. When in directory format, the library will occupy a disc file of only 640 blocks.

SEMI-COMPILED OUTPUT FILE

A file is required to receive semi-compiled output from the compiler. Any previously allocated permanent file may be used if specified by the programmer in a **SEND TO** statement. If a **SEND TO** statement is not provided, the compiler assumes the equivalent of:

SEND TO (ED, ICLA-DEFAULT. FORTSEMICOMP)

(see *The SEND TO statement*, page 21). In other words, a default file with name ICLA-DEFAULT must be made available to receive the semi-compiled output.

Other compilers may also require the use of an ICLA-DEFAULT file. To allow for multiprogramming, further files with this name but with different generation numbers may be allocated; the compiler chooses the file with the highest generation number.

A **SEND TO** statement is always required when output is to an F.D.S. file. The above facility can be obtained by specifying the statement:

SEND TO (FD)

in which case the compiler assumes the equivalent of:

SEND TO (FD, ICLA-DEFAULT. FORTSEMICOMP)

The file size required depends entirely on the program being compiled. Compiled programs of length less than 32K words should be accommodated in a file of length 640 buckets (eight cylinders), but many programs will occupy considerably less space than this. The number of buckets used in any compilation is given at the end of the compiler listing on the line printer.

If the file is not large enough, the compiler will attempt to extend it on the same cartridge. Any extension made will also be reported at the end of the compiler listing on the line printer. If there is insufficient room for the extension on that cartridge, the program will halt, and the following message will be output:

0#XFAE; HALTED:- Z1

If this happens, the run must be abandoned (see *Other exception conditions*, page 117).

WORK FILE

During consolidation, **#XPCK** requires work space on disc. A previously allocated permanent file may be used for this purpose if specified by the user in a **WORK** statement in the program description. See the section *The WORK statement*, page 22. If a **WORK** statement is not specified, a scratch file of 80 blocks will be opened.

Optional disc files

PROGRAM DUMP FILE

If a binary dump of an object program is required on a disc file following consolidation, this file must have been previously allocated and must have a name of the form **PROGRAMVzzzz** where **zzzz** are any four characters starting with a letter; such a name will ensure that the operator can load the program into store by means of a **FI** message. Object program disc files may also be used if required.

Efficiency considerations with E.D.S.

The 1900 FORTRAN 16K disc compiling system requires at least one E.D.S. transport, but a much more efficient system is achieved with a minimum of two transports.

Assuming two E.D.S. transports are available, **#XFAE**, **#XPCK** and the **SRF7** library group should be as near as possible to each other on one E.D.S. cartridge. Compiler semi-compiled output files should be on the other E.D.S. cartridge.

For those consolidations producing a binary dump on disc, the optimum position of the binary output file for efficiency will depend on various factors, e.g. the number of input files, the size of the program, the number of E.D.S. drives etc. In general it will be more efficient to have the binary output file on a different E.D.S. cartridge from the semi-compiled file.

Object programs using E.D.S. files will vary considerably with circumstances. The one general point is that files should be arranged to minimise E.D.S. head movement, the principle for all E.D.S. file arrangement.

In general for all programs which are to be loaded into store by means of a find process, the file containing the program should be as near to the file **XPEBOOTSTRAP** (see specification of **#XPEB**) as possible, or on a separate E.D.S. cartridge.

USE OF GEORGE

Batch Development System

The way in which the Batch Development System is used depends on the installation configuration and organization. Programmers will normally submit work for batch compilations to a work assembler whose duties are to:

- 1 Collect together all jobs that reference programs in the same file, verifying that each job has been prepared correctly.
- 2 Supply the appropriate job description, document header and terminator for each batch of jobs formed, and submit the batch to be run.
- 3 Inform the operator of the input file and any other files that are likely to be required.
- 4 Decollate the batch after it is run, and return the jobs to the programmers.

Chapter 14 Operator's guide to the FORTRAN disc compiler #XFAE

This chapter describes the operating instructions for the FORTRAN disc compiler #XFAE. Although written as a reference document for operators, it is also intended as a programmer's guide for writing the operator's instructions. Instructions for running the editor, #XMED, are also included.

The operating instructions are divided into two parts: the first lists the normal procedure and the second lists exception conditions that can arise. Each section is further divided into compilation instructions and run time instructions.

Operating instructions for running under GEORGE 1 and 2 are given in the appropriate *GEORGE 1 and 2* manual for each system. Some information on running #XFAE under GEORGE is given at the end of this chapter.

HARDWARE REQUIREMENTS

Compiler and consolidator

Processor	Any 1900 Series central processor with floating point facilities, performed either by extracode or hardware, and at least 10,240 words of core store.
Basic peripherals	At least one reader, paper tape or card. Optional listing device, a line printer.
Discs	At least one E.D.S. transport (preferably two) or at least one F.D.S. transport. The disc transport(s) will hold the compiler, consolidator and library, in addition to the semi-compiled output file, and an optional binary dump file. Other disc files may be optionally used to hold source program.
Magnetic tapes	Magnetic tape may optionally be used to hold source programs.

Editor

Storage	6016 words of core store.
Basic peripherals	1 paper tape reader and/or 1 card reader. 1 line printer with standard control loop (channel 1 = top of page; channel 8 = bottom of page). 1 paper tape punch (optional). 1 card punch (optional).
Discs	1 E.D.S. transport and/or 1 Twin E.D.S. transport and/or 1 F.D.S. unit. Further units may optionally be allocated.
Magnetic tapes	Magnetic tapes may optionally be used to hold source programs.

USE OF PERIPHERALS

Compiler #XFAE

LP0	Listing peripheral
TR0	Paper tape input
CR0	Card input
ED0 } or FD0 }	FORTRAN compiler file containing the compiler. This file remains assigned throughout the compilation, since the compiler is overlaid

ED1 } or FD1 }	Disc file nominated to receive semi-compiled output
----------------------	---

ED2 Input E.D.S. file

FD2 Input F.D.S. file

MT2 Input magnetic tape

Consolidator #XPCK (mark 11C and later)

(The consolidator is loaded and entered automatically)

LP0 Listing peripheral

ED1 to ED12 } or FD1 to FD12 }	Used for input disc files, including semi-compiled and library files. Numbers are allocated in the same order as the files are accessed, and each file is given a unique number
--------------------------------------	--

ED13 } or FD13 }	Consolidator work file
------------------------	------------------------

ED14 } or FD14 }	Optional file for binary output
------------------------	---------------------------------

If a file is used for more than one purpose, the lowest appropriate number is allocated. The two output files (numbers 13 and 14) must not, however, share the same file.

Editor #XMED

LP0 Listing peripheral

TR0 Paper tape reader for input of parameters and amendments

CR0 Card reader for input of parameters and amendments

TP0 Paper tape punch, used for subfile copies (optional)

CP0 Card punch, used for subfile copies (optional)

ED1 } or FD1 }	Input disc file
----------------------	-----------------

ED2 } or FD2 }	Output disc file
----------------------	------------------

Object program

The object program will use those peripherals specified by the user, the associated unit numbers being likewise specified by the user. The peripherals will be assigned when required and released at the end of the program, unless the user specifically releases them during the run.

ED0 } or FD0 }	For overlaid object programs, this file will be retained for overlaying purposes.
----------------------	---

PRIORITY

The compiler as supplied has a priority of 73.

The consolidator as supplied has a priority of 60.

The editor as supplied has a priority of 50.

OPERATING INSTRUCTIONS

Compilation and consolidation

Narrative

Console message

- 1 To load the compiler
either
 - (a) input
if #XFAE is held in a disc file with name
PROGRAMVyyyy or,
FI #XFAE #yyyy
 - (b) input
if #XFAE is held in a disc file with name
PROGRAMVXFAE
FI #XFAE

A disc or discs with a compiler file (containing the compiler and the consolidator #XPCK) and a file SUBGROUPSRF7 containing the SRF7 library must be on-line. The library file must be on the same medium as the compiler file. When the compiler has been loaded, the message is output

0 #XFAE;HALTED:-LD
- 2 Activate the compiler by one of the following:
 - (a) begin, reading initially from paper tape
GO #XFAE 20
 - (b) begin, reading initially from cards in
1900 code
GO #XFAE 21
 - (c) begin, reading initially from cards in
Atlas code
GO #XFAE 22
- 3 The source program will be read in and compiled. Switching of readers may occur. At the end of compilation one of the following events will occur.
 - (a) If a group of segments (not a complete program) is being compiled or the program is incomplete in some other respect, the compiler will output
To initiate another compilation return to Operating Instruction 2.
0 #XFAE;HALTED:-EC
 - (b) If a complete program has been successfully compiled the compiler outputs
#XPCK is automatically entered, and starts consolidating.
0 #XFAE;DELETED:-FI #XPCK #yyyy
- 4 At the end of consolidation
 - (a) If the object program is to be loaded into store, the consolidator #XPCK renames itself #name, which is the object program name. When the object program is in store, ready to be run, the message is output
0 #name;HALTED:-LD
 - (b) If the program is to be written to the disc in binary form, the consolidator outputs the binary program and terminates with the message
0#XPCK;DELETED:-HH

Loading and running the object program

Narrative

Console message

- 5 Load the object program, if this has not been left in store, either

Narrative

- (a) by inputting
if the binary program is held in a file with
name PROGRAMVyyyy
or
 - (b) by inputting
if the binary program is held in a file
with name PROGRAMVname
When the object program has been
loaded the message
will be output
- 6 Activate the object program as follows:
- (a) if no TRACE output is required (the
normal case) input
 - (b) if TRACE output is required for every
statement, input
 - (c) if a TRACE steering list is supplied, input
Load the steering list on the basic
peripheral assigned. The steering list will be
read in, and the program entered.
- 7 On request from the programmer:
TRACE output may be suppressed by turning
switch 0 off by inputting
TRACE output may be restarted by inputting
- 8 The program will delete itself with the
message
where *abcde* are any five octal digits defined by the
programmer.

Console message

FI #name #yyyy

FI #name

0#name;HALTED:—LD

GO #name 20

GO #name 27

GO #name 28

OFF #name 0

ON #name 0

0#name;DISPLAY:—*abcde*
DELETED:—00

Loading and running the editor, #XMED

Narrative

- 1 Load #XMED
- 2 Load the direct access cartridges
- 3 Load the editing file in the card or paper tape reader
- 4 Either
 - (a) for input starting on paper tape, input
 - or
 - (b) for input starting on cards in 1900 code,
input
 - or
 - (c) for input starting on cards in ATLAS code,
input
- 5 At the end of a run, either
 - (a) #XMED will halt with the message
in which case it may be restarted
for another run at step 2,
or
 - (b) #XMED will delete itself with the message
where *name* and *file* are as specified in
the *LOAD directive (see page 53).

Console message

GO #XMED 20

GO #XMED 21

GO #XMED 22

#XMED;HALTED:—OK

#XMED;DELETED:—FI #name #file

DISC ERROR CONDITIONS

When using disc files (both on E.D.S. and F.D.S.) there are certain circumstances which can prevent the successful completion of a disc operation. These can occur in the compiler, consolidator, or in object programs which use disc files. The messages generated by these conditions and the actions to be taken are given below. In the messages *name* may be either XFAE, XPCCK or the object program name.

<i>Message</i>	<i>Reason</i>	<i>Action</i>
1 0# <i>name</i> ;HALTED:—NEED ED } FD } <i>f,g</i>	The named file is not in the system	Load the correct cartridge and input GO # <i>name</i> or abandon run.
2 0# <i>name</i> ;HALTED:— SYSTEM OVERLOAD	Either (a) system file open for writing when trying to open another file or (b) insufficient core store for holding further file descriptions or (c) an attempt has been made to open more than the maximum permitted number of files.	To continue, wait until the condition has cleared and input GO # <i>name</i> or abandon run. (In the case of an object program input GO # <i>name</i> 29)
3 0# <i>name</i> ;HALTED:— C3 }	Operator has input GO after a FAIL or ERR message for a disc file	Abandon run. (In the case of an object program input GO # <i>name</i> 29)
4 0# <i>name</i> ;HALTED:— K1 }		

The following error conditions occur only during consolidation; *name* will be either XPCCK or the object program name

<i>Message</i>	<i>Reason</i>	<i>Action</i>
5 0# <i>name</i> ;HALTED:— G2	Failure of integrity code check.	Abandon run, or wait until the file needed has been freed by another program and then input GO # <i>name</i>
6 0# <i>name</i> ;HALTED:— RL	A file previously used by the consolidator has been removed from the system.	Make file available and input GO # <i>name</i>

OTHER EXCEPTION CONDITIONS

Compilation

<i>Message</i>	<i>Reason</i>	<i>Action</i>
1 0#XFAE;HALTED:— ZZ	Program has compilation error(s).	Abandon run. To initiate another compilation return to Operating Instruction 2.

<i>Message</i>	<i>Reason</i>	<i>Action</i>
2 0#XFAE;HALTED:— CE	Checksum error when reading semi-compiled program.	If input is on paper tape or cards, the tape should be moved back to the beginning of a block or one card moved back to be re-read. Continue by inputting GO #XFAE If semi-compiled program was being read from disc or magnetic tape, abandon the compilation by inputting GO #XFAE 27 The compilation in error should be repeated from the beginning, possibly after the regeneration of the semi-compiled program.
3 0#XFAE;HALTED:—CR LP TR TP	The peripheral required by the compiler is not available.	Make the necessary peripheral available and input GO #XFAE
4 0#XFAE;HALTED:— PF	This message occurs after a compilation has been abandoned by inputting GO #XFAE 27	To initiate another compilation, return to Operating Instruction 2.
5 0#XFAE;HALTED:— ST	The compiler requires more core store and none is available.	(a) On a multi-programming machine, delete an idle program and continue by inputting GO #XFAE or abandon the compilation as in (b). (b) For a single programming machine, abandon the compilation by inputting GO #XFAE 27. The program is too large and should either be compiled on a machine with more core or be rewritten in smaller segments.
6 Operator action is required if the compiler attempts to read from a slow device more input than the user has supplied.	If the message in Operating Instruction 3 has not been output, it is likely that a FINISH statement has been omitted.	Simulate a FINISH statement by inputting GO #XFAE 28
7 0#XFAE;HALTED:—ER7	Compiler held on magnetic tape.	Transfer the compiler to disc.
8 0#XFAE;HALTED:— Z1	Insufficient room for extension of semi-compiled output file.	Abandon run.

Consolidation

Message

1 0#XPCK;DISPLAY:— MO

Reason

Program is incomplete.

Action

Consolidation continues without the missing segments.

2 0#XPCK;HALTED:— EM

Attempt to load extended data program on a processor without extended mode capability.

Abandon run. Recompile with correct compiling statements.

3 0#XPCK;HALTED:— ER

Apparent error in semi-compiled data.

Abandon run. Attempt consolidation again with card or paper tape parameters. If error persists, recompile.

4 0#XPCK;HALTED:— LP

The line printer required by the consolidator is not available.

Make the necessary peripheral available and input
GO #XPCK.

5 0#XPCK;HALTED:— PX

Error in information passed by compiler.

Abandon run. Either repeat compilation or repeat consolidation with parameters on paper or cards.

6 0#XPCK;HALTED:— ST

The consolidator requires more store.

Make more store available and input
GO #XPCK.

7 0#XPCK;HALTED:— YY

Reason for the halt is printed on the line printer. Normally the message will indicate a user error but for those messages given below operator action can be taken.

8 0#XPCK;HALTED:— CE

Checksum error detected in sub-routine library directory.

Abandon run and recreate the subroutine library.

9 0#XPCK;HALTED:— ND

Subroutine library in directory format used with #XPCK/10C or earlier version.

Abandon run and run again using later consolidator.

Line printer message

WORK FILE NOT OPENED: SYSTEM CONTROL
AREA FULL
WORK FILE NOT OPENED:
INSUFFICIENT SPACE AVAILABLE

Action

Rectify condition if possible and input
GO #XPCK

Loading and running the object program

Message

1 0#name;HALTED:— EE

Reason

The program run has terminated. This message occurs either on the detection of an execution error or as a result of a
GO #name 29

Action

No operator action required. Post mortem information will be output automatically. The program cannot be restarted.

2 0#name;HALTED:— abcde

The program has reached the statement PAUSE abcde whose abcde consists of up to five octal digits.

As specified by the programmer.

<i>Message</i>	<i>Reason</i>	<i>Action</i>
3 0#name;HALTED:— CR CP LP TR TP	The peripheral required by the program is not available.	Make the necessary peripheral available and input GO #name
4 0#name ILLEGAL	Illegal occurrence.	Input GO #name 29 Post mortem information will be output and the program will halt with the message 0#name;HALTED:— EE
5 0#name;HALTED:— OVRL ER1	The program has become corrupted (e.g. by array subscript error).	Abandon run.
6 0#name;HALTED:—OVRL ER5	A disc transfer has failed while bringing down an overlay.	Abandon run.
7 0#name;HALTED:—OVRL ER6	Overlaid binary program has become corrupted.	Abandon run.
8	The program loops indefinitely.	Input GO #name 29 Post mortem information will be output and the program will halt with the message 0#name;HALTED:— EE

Loading and running the editor, #XMED

<i>Message</i>	<i>Reason</i>	<i>Action</i>
1 #XMED;HALTED:— CR TR CP TP LP	The peripheral required by the editor is not available.	Make the necessary peripheral available and input GO #XMED
2 #XMED;HALTED:— ED	E.D.S. or twin E.D.S. file not found in the system. Either the file is not on-line, or else its name is incorrectly specified.	If the file is not on-line, make it available and input GO #XMED Otherwise abandon the run.
3 #XMED;HALTED:— FD	F.D.S. file not found in the system. Either the file is not on-line or else its name is incorrectly specified.	If the file is not on-line make it available and input GO #XMED Otherwise abandon the run.
4 #XMED;HALTED:— IC	Failure of integrity code check: the direct access file is being used by another program.	Wait until the other program is deleted, and then input GO #XMED
5 #XMED;HALTED:— UT	A utility program is writing to the system control area.	Wait until this utility program is deleted, and then input GO #XMED
6 #XMED;HALTED:— EE	Executive cannot handle any more E.D.S. or Twin E.D.S. files.	Wait until one of the programs using E.D.S. or Twin E.D.S. files is deleted, and then input GO #XMED

<i>Message</i>	<i>Reason</i>	<i>Action</i>
7 #XMED;HALTED:– FF	Executive cannot handle any more F.D.S. files.	Wait until one of the programs using F.D.S. files is deleted, and then input GO #XMED
8 #XMED;HALTED:– PA	A parity error has occurred in reading a bucket from the direct access file.	To ignore the bucket, input GO #XMED
9 #XMED;HALTED:– TE	A transfer error has occurred in reading from cards or paper tape.	To ignore the card or line, input GO #XMED
10 #XMED;HALTED:– ER	The direct access output file has been filled.	The program may be restarted at operating instruction 2 for a further run.
11 #XMED;HALTED:– ZZ	The run has been abandoned by the program. The reason is given on the listing.	The program may be restarted at operating instruction 2 for a further run.

RUNNING #XFAE UNDER GEORGE

Batch Development System

Initially the operator will load the required files and cause the job description to be read in. Normally no further intervention will be required.

Messages may occur asking for other discs or magnetic tapes to be loaded. The operator should load these and cause the job to continue. All output produced should be returned to the work assembler (see *Use of GEORGE*, page 112).

If for some reason, such as looping, it is necessary to abandon the current program and proceed with the next one in the batch, the operator may type the message:

GO #george 26

If the whole batch has to be abandoned, the message:

GO #george 25

will cause the next GEORGE job to be initiated. In either case, *george* represents the appropriate name of GEORGE 1 or 2 (e.g. XKEE: see the ICL 1900 Series manual *Operating Systems GEORGE 1 and 2*).

If the batch is restarted at a later time, processing will continue at the program following the one which was being processed when the run was abandoned.

Appendix 1 Compilation/consolidation errors

COMPILATION ERROR NUMBERS

In the following table of error numbers, errors 1 to 25 and 36 to 46 are interpreted regardless of statement type. The significance of errors 26 to 29 depends on the statement type.

<i>Error number</i>	<i>Interpretation</i>
1	Mis-matched parenthesis.
2	Missing parenthesis.
3	/), expected but another character found.
4	Error in format of expression.
5	Constant out of range.
6	Integer variable expected, but something else found.
7	Array not declared, or declared twice.
8	First character of name non-alphabetic, or name omitted, or array or function has been omitted.
9	Statement too long.
10	Statement incomplete.
11	Label set twice or error in columns 1 to 5.
12	Error in label references.
13	Incorrect subscript e.g. too many or too few subscripts.
14	Incorrect name e.g. subroutine has incorrect name.
15	DO label not found.
16	Label missing.
17	Statement not recognised or out of context.
18	Error in compiling system statement.
19	Either expression too complicated or too many overlay statements (as a guide, this may occur if more than 150 segments are specified in OVERLAY statements in the program description).
20	Compiler error (should not occur)
21	Expression on left hand side of arithmetic assignment statement, or “=” has been used instead of “EQ.” in a relation expression.
22	Array not declared in a declaration statement or Statement Function definition out of place.
23	The length of a COMMON block has exceeded the limit.
24	Too many initial statements declared.
25	Reserved.

<i>Error number</i>	<i>Statement</i>	<i>Interpretation</i>
26	Initial statements	Incompatible use of file or subfile names e.g. file specified in SEND TO statement is same as file specified in DUMP ON statement.
	CALL	Subroutine called has the same name as the segment containing the CALL statement.
	FUNCTION	Segment name omitted.
	Function reference	Function referenced has the same name as the master or subroutine segment which contains the function referenced.
	SUBROUTINE	Segment name omitted.
	GO TO	Error in GO TO statement following the characters GO TO.
	DO	DO statement incomplete.
	IF	DO statement in 'logical IF'.
	FORMAT	Unlabelled.
	READ } WRITE }	Channel number not integer constant or integer variable.
	INTEGER REAL LOGICAL DOUBLE PRECISION COMPLEX }	Type already specified for same item or type statement in wrong position.
	COMMON	Incorrect name for common block.
	DIMENSION	No dimensions given.
	EQUIVALENCE	Dummy argument in Equivalence list.
	DATA	Incorrect name in list e.g. a subroutine name.
27	FUNCTION	No arguments after function name.
	Function reference	An argument of the function has the same name as the master or subroutine segment in which the function is referenced.
	GO TO	Error after) of 'computed GO TO'
	CALL	An argument of the subroutine called has the same name as the master or subroutine segment in which the CALL statement appears.
	READ } WRITE }	Format reference error.
	COMMON	Incorrect item in list e.g. dummy argument.
	EQUIVALENCE	Incorrect item e.g. constant or segment.
28	DATA	Incorrect form of constant.
	DO	Parameters not integer expressions.
	READ } WRITE }	Error in list.
	EQUIVALENCE	Array in list not declared as such.
29	DATA	Incorrect form of complex constant.
	EQUIVALENCE	Item already equivalenced or otherwise defined so that equivalence is impossible.
	DATA	Too many or too few constants in list.

<i>Error number</i>	<i>Interpretation</i>
36	Incomplete program. A source program description segment has been omitted or is not the first segment input.
37	Reserved.
38	Subfile of the correct name and containing FORTRAN source or semi-compiled program has not been found.
39	File name has been omitted from the first READ FROM statement for magnetic tape.
41	A block has been read from magnetic tape containing more than 128 words.
43	A source line in a disc file is too long. Some following lines may be ignored.
44	A bucket in the disc file contains more than 128 words of information. The first 128 are accepted.
45	Failure of integrity code check.
46	Purge date not exceeded for E.D.S. file.

CONSOLIDATION ERROR MESSAGES

Certain error conditions may arise during compilation which will cause the compilation/consolidation process to terminate and one of the following messages to be output after the listing.

<i>Line printer message</i>	<i>Action</i>
INSUFFICIENT SPACE IN WORK FILE	Re-run with larger work file.
INSUFFICIENT SPACE IN OUTPUT FILE	Re-run with larger output file.
<i>f(g)</i> PURGE DATE NOT EXCEEDED	Abandon run.
LOWER DATA > 4096 WORDS	Abandon run.
COMPACT PROGRAM > 32768 WORDS	Abandon run.
START OF UPPER DATA > 32768	Abandon run.
WORK FILE NOT OPENED: SYSTEM CONTROL AREA FULL	Rectify condition if possible and input GO #XPCK
WORK FILE NOT OPENED: INSUFFICIENT SPACE AVAILABLE	Rectify condition if possible and input GO #XPCK

The following line printer messages report non-fatal errors during consolidation.

SEGMENTS MISSING

INCOMPATIBLE MODE: SEGMENT *nnnnn*

Appendix 2 Execution error numbers

The following table gives the interpretation of run time error numbers. Those marked with an asterisk * are fatal. Non-fatal errors may be trapped and the program continued if desired (see *The error trap facility*, page 35).

Specifications of all the subroutines cited in this appendix may be found in the manual *FORTRAN: Compiler Libraries*.

<i>Error number</i>	<i>Interpretation</i>
0'x'	This error number indicates that a prohibited character x has been found in an input field.
01	Array subscript error. The generated address of an array-element is outside the bounds of the array. This error check is made only when the program is compiled in TRACE mode. Alternatively, when using the subroutine FMOVE, the number of elements to be transferred exceeds the size of the array specified by the second argument. (This check cannot be made if an array element is specified.)
*02	The TRACE steering-list has incorrect format or is too long.
*03	The overlay depth has been exceeded.
04	Reserved.
*05	Number of arguments in any of AMAX0, AMAX1, AMIN0, AMIN1, DMAX1, DMIN1, MAX0, MAX1, MIN0, MIN1 is less than 2.
06	An attempt has been made to read beyond end of data in the Batch Development System.
07	Not enough arguments have been supplied for the subroutine called: This error is detected only when the object program is an overlay program.
*08	This error can occur only in mixed-language programs. Too many arguments have been specified for a PLAN segment.
09	Reserved.
*10	Error in the structure of a format specification, e.g. field separator missing, character out of place, unrecognised character found.
*11	'F' or 'E' conversion code is not associated with a REAL or COMPLEX variable.
*12	'I' conversion code is not associated with an INTEGER variable.
*13	'G' conversion code is not associated with a REAL or COMPLEX variable.
*14	'L' conversion code is not associated with a LOGICAL variable.
*15	'D' conversion code is not associated with a DOUBLE PRECISION variable.
*16	Zero or undefined field width when using the A,L,H or X codes.
*17	When using the 'L' code, either whole of the input-field is blank, or the first non-space character is neither T nor F.
18, 19	Reserved.
*20	The channel number used in the program has not been associated with a peripheral in the program description.
*21	A formatted input record is not long enough.
*22	An unformatted input record is not long enough.
*23	Illegal operation attempted, e.g. read from card punch, or write to an input magnetic tape.
24-26	Reserved.

<i>Error number</i>	<i>Interpretation</i>
*27	An attempt has been made to use RENAME with an input file.
*28	An attempt has been made to use FILE or RENAME but a dummy filename has not been given in the program description statement.
*29	An attempt has been made to use FILE when the file is already open.
*30	Illegal sequence of operations on serial file, e.g., read after write, read or write after ENDFILE.
*31	No filename specified for input device.
*32	Input file has incorrect format, e.g. no start-of-data sentinel found when opening input magnetic tape.
33	Reserved.
*34	End of tape mark detected when writing to magnetic tape.
*35	Failure of data block-count check (unformatted magnetic tape).
*36	An attempt has been made to write a formatted record longer than the maximum permitted block or bucket size.
*37	Block-size on input file exceeds buffer size specified in the program description.
38	Reserved.
*39	An attempt has been made to read a formatted record longer than the available buffer or the bucket size in an unformatted file exceeds the buffer size.
40	Reserved.
*41	Failure of integrity-code check on opening E.D.S./F.D.S. files.
*42	Purge-date not exceeded on opening output E.D.S./F.D.S. files.
*43	For E.D.S. & F.D.S. On: opening a scratch-file extending a file renaming a file } system control-area full.
*44	On: opening a scratch-file extending a file renaming a file } no space available.
*46	On: opening a scratch-file extending a file renaming a file } Specified file not opened as a write file. (Should occur only when renaming)
*47	Argument outside the permitted range in FERROR.
48 & 49	Reserved.
50	Overflow has been set.
*51	FTRAP used with TRACE 0.
*52	FRESET used with TRACE 0.
53	Result exceeds capacity in EXP.
54	Result exceeds capacity in DEXP.
55	Non positive argument in ALOG.
56	Non positive argument in DLOG.
57	Negative argument in SQRT.
58	Negative argument in DSQRT.
59	Attempt to form ATAN2 (0/0).
60	Attempt to form DATAN2 (0/0).

<i>Error number</i>	<i>Interpretation</i>
61	Attempt to form ISIGN (x, 0).
62	Attempt to form SIGN (x, 0).
63	Attempt to form DSIGN (x, 0).
64	Attempt to find MOD (x, 0).
65	Attempt to find AMOD (x, 0).
66	Attempt to find DMOD (x, 0).
67	Argument < 1 in ACOSH
68	$ \text{Argument} > 1$ in ATANH
69	$ \text{Argument} \leq 1$ in ACOTH
70	$ \text{Argument} > 1$ in ASIN
71	$ \text{Argument} > 1$ in ACOS
72	Result exceeds capacity in TAN
73	Result exceeds capacity in CABS
74	Results exceeds capacity in CSIN.
75	Results exceed capacity in CCOS.
76	Argument too large to be held as an integer in NINT.
77	Result too large to be held as an integer in IFIX.
78	Result too large to be held as an integer in INT.
79	Trace list too short.
*80	Error in the information block used by GENAH
81	The array specified as the third argument DEFBUF is not large enough to contain the number of characters specified in the second argument.
82	Semi-compiled segments generated by #XFAM/2C and earlier versions, or #XFAP/1, #XFAC/1 compilers have been included in object programs produced by #XFAM/4, #XFAE/3, #XFAT/1, or later versions of these compilers.
83	Illegal use of the subroutine FMOVE, e.g. a call has been made to this subroutine from a FORTRAN segment produced by #XFAM/4, #XFAE/3 or earlier versions of these compilers.
84	The number of elements transferred by the subroutine FMOVE is less than zero.
85 to 97	Reserved.
98	Misuse of monitor peripheral (e.g. BACKSPACE on channel 0).
99	Reserved.
*100	The object program has reached an END statement.
101 to 103	Reserved.
104	An attempt has been made to enter a missing segment.
105	An attempt has been made to read beyond the end of data on a card input channel.
106	An attempt has been made to read beyond the end of data on a paper tape input channel.
*107	The end of file on magnetic tape input has been encountered.
*108	The end of file on serial disc (E.D.S. or F.D.S.) has been encountered.
109 to 113	Reserved.
*114	A CALL, with arguments, to an overlay segment has caused the calling segment to be overwritten.

<i>Error number</i>	<i>Interpretation</i>
-------------------------	-----------------------

115 to 120	Reserved.
------------	-----------

*121	An attempt has been made to write more than one record, or to write too long a record to a DEFBUF array in a single WRITE.
------	--

122 to 499	Reserved.
------------	-----------

Error numbers 500 to 515 refer to files on E.D.S. and F.D.S. used by the Backing Store Package (see the manual *FORTTRAN; Compiler Libraries*).

*500	Insufficient space for reading from or writing to a file (i.e. K specified incorrectly).
------	--

*501	Failure of integrity code check when opening a file.
------	--

*502	Purge date not exceeded on an output file.
------	--

*503	Logical bucket number out of range (i.e. program corrupted).
------	--

*504	System control area full.
------	---------------------------

*505	No space available for extending a file or opening a scratch file.
------	--

*506	File does not have one-block buckets.
------	---------------------------------------

*507	Attempt to write to a file opened by INFILE.
------	--

*508	Undeclared channel number used.
------	---------------------------------

*509	More than 8 channel numbers declared.
------	---------------------------------------

*510	Incorrect format of parameters in opening procedure.
------	--

*511	Elements specified in GETPART or PUTPART are not within bounds, or are incorrectly related.
------	---

*512	Channel number is not in the range $1 \leq k \leq 15$.
------	---

*513	Not enough space available for scratch file (or file extension).
------	--

*514	Channel number already in use.
------	--------------------------------

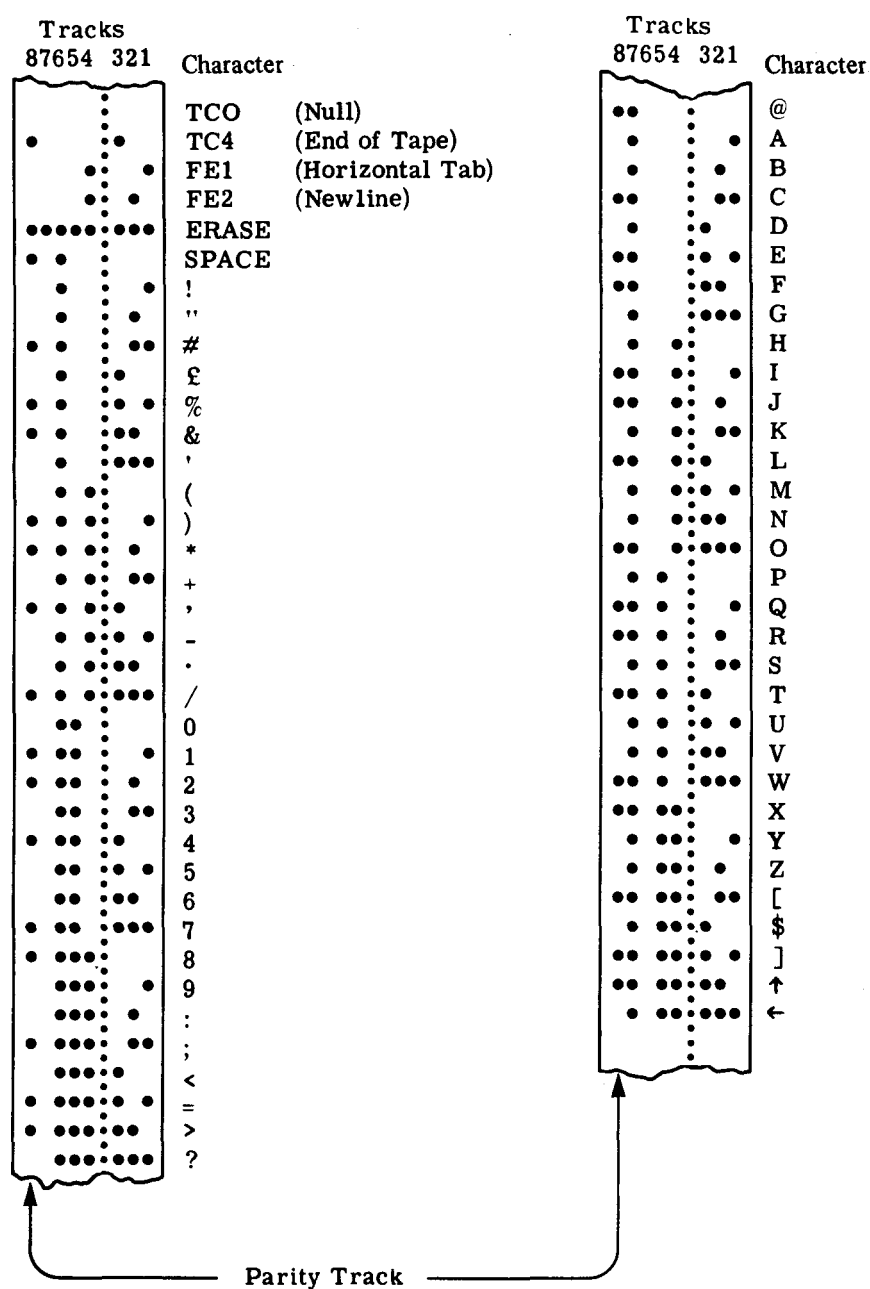
*515	Unit number already in use.
------	-----------------------------

Error numbers 900 to 998 and *999 are available to the user for use with the subroutine FERROR.

Appendix 3 Paper tape and card codes

This appendix lists all the characters used for FORTRAN programs and data.

PAPER TAPE CODE



1900 CARD CODE

<i>Character</i>	<i>Card punching</i>
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
space	NONE
&	10 or 10/0
#	3/8
@	4/8
(	5/8
)	6/8
]	7/8
A	10/1
B	10/2
C	10/3
D	10/4
E	10/5

<i>Character</i>	<i>Card punching</i>
F	10/6
G	10/7
H	10/8
I	10/9
J	11/1
K	11/2
L	11/3
M	11/4
N	11/5
O	11/6
P	11/7
Q	11/8
R	11/9
S	0/2
T	0/3
U	0/4
V	0/5
W	0/6
X	0/7
Y	0/8
Z	0/9

<i>Character</i>	<i>Card punching</i>
-	11
"	11/0
/	0/1
+	10/2/8
.	10/3/8
;	10/4/8
:	10/5/8
'	10/6/8
!	10/7/8
[	11/2/8
\$	11/3/8
*	11/4/8
>	11/5/8
<	11/6/8
↑	11/7/8
£	0/2/8
,	0/3/8
%	0/4/8
?	0/5/8
=	0/6/8
←	0/7/8

ATLAS CARD CODE

<i>ATLAS character</i>	<i>Card punching</i>	<i>1900 FORTRAN interpretation</i>	<i>ATLAS character</i>	<i>Card punching</i>	<i>1900 FORTRAN interpretation</i>
0 TO 9	As 1900 code	0 to 9	.	10/3/8	.
A to Z	As 1900 code	A to Z	)	10/4/8	)
space	none	space	>	10/5/8	>
=	3/8	=	— (underline)	10/6/8	@
'	4/8	'	(vertical bar)	10/7/8	
&	5/8	&	π	11/3/8	\$
:	6/8	:	*	11/4/8	*
document terminator	7/8	]	?	11/5/8	?
+	10	+	]	11/6/8	]
-	11	-	[	11/7/8	[
/	0/1	/	,	0/3/8	,
			(	0/4/8	(
			<	0/5/8	<

Appendix 4 Data formats on disc files

FORMATTED FILES

Formatted files are held in a subset of the Direct Access Housekeeping formats.

Bucket format

Each bucket will have words 0 and 1 as a standard bucket header.

Word 0		0
Word 1	bits 0 to 11	The minimum buffer size needed for a disc file. (See <i>Buffer size</i> , page 12.)
	bits 12 to 23	Number of useful words of information, including header.
Word 2	record 0	
onwards	record 1	
	etc.	

End of file bucket

Word 0	bits 0 to 12	0
	bit 13	End of file marker (only checked on input).
	bits 14 to 23	0
Word 1	bits 0 to 11	Buffer size needed for the file.
	bits 12 to 23	2
Word 2		0
onwards		

Record format

Word 0	bits 0 to 11	0
	bits 12 to 23	Count of number of words in record.
Word 1		Words of record information.
onwards		

No specific allowance is made for print control characters. Hence, any file which may subsequently be off-lined should be created with the first character of each record as a print control character which can be interpreted by an off-lining routine.

Note: For formatted files a buffer size can be specified in the relevant program description statements. If no buffer size is specified, a buffer size of 128 words is assumed.

UNFORMATTED FILES

Data bucket

Each unformatted record consists of one or more buckets of data. No block may contain more than one unformatted record. The first 5 words of each bucket are reserved for the following information

Bucket header	{	Word 0	0	
		Word 1 bits 0 to 11	The minimum buffer size needed for the file.	
		bits 12 to 23	Number of words in bucket actually used, including header.	
Start of bucket	{	Word 2	Number of words in bucket actually used, <i>excluding header</i> .	
		Word 3	Reserved	
		Word 4 bit 0	{	
			1	If this is last bucket of the record.
			0	Otherwise.
		bits 1 to 23		Bucket number relative to the start of the record.

The actual data in each bucket begins at word 5.

End of file bucket

Word 0	bits 0 to 12	0
	bit 13	End of file marker (only checked on input).
	bits 14 to 23	0
Word 1	bits 0 to 11	Buffer size needed for the file.
	bits 12 to 23	5
Words 2 to 4		As for normal bucket, ignored on input.

Rest of the bucket zero.

Note: For unformatted files a buffer size can be specified in the relevant program description statements. If no buffer size is specified, the buffer size is assumed to be the maximum buffer size already specified in the program description for an *unformatted* file, unless this maximum is less than 128, in which case the buffer size is assumed to be 128 words.

Appendix 5 Data formats on magnetic tape files

The format of FORTRAN formatted files on magnetic tape is the same as that used for EMA and Algol. Unformatted files are intended for use only in FORTRAN programs. Both types of files may be created and read by the 1900 Series Magnetic Tape Housekeeping Package.

FILE STRUCTURE

Any FORTRAN magnetic tape file, either formatted or unformatted, consists of the following items in the order shown:

- Header label
- Start of data sentinel
- One or more data blocks
- Trailer label

Only single reel files are allowed. Start of data sentinels and trailer labels are in standard 1900 format.

FORMATTED DATA BLOCKS

A formatted block consists of one or more records, each record having a format as follows:

- Word 0 Number of words in the record.
- Word 1 Carriage control character in the least significant position. This will normally have the value #41 although the first record in a file may have the value #51 in this position.
- Word 2 Record information space filled to give an integral number of words.
onwards

Notes

- 1 A record may not be split over two blocks.
- 2 A minimum block size of five words is always output, and zero words will be added to produce this minimum block size where necessary.

UNFORMATTED DATA BLOCKS

An unformatted data block may contain all or part of one record only, having the following format:

- Word 0 Number of words in the block.
- Word 1 Data block sequence number relative to the start of the tape, starting at 1.
- Word 2 Data block sequence number relative to start of record, starting at record 1.
- Word 3 Number of words of information following. The sign bit of this word is set in the last block of each record.
- Word 4 Record information
onwards

If a record output to an unformatted file is larger than the size of buffer specified, then the record will be split over two or more blocks. If the record is smaller, then a block of sufficient size to hold the record will be output. Note that each block of a FORTRAN unformatted record may be treated as a complete record if the file is processed by the Magnetic Tape Housekeeping Package.

Index

*1900 directive, the	54	Common storage areas	64
1900 card code	132	Communication of data between PLAN and FORTRAN segments	57
Allocation of files	110	Compact data programs	47
*ALTER directive, the	55	Compilation/consolidation process	21, 24, 109
AMENDBATCH macro, the	91	Compilation error diagnostics	33
AMENDBATCHMT macro, the	93	Compilation error numbers	123-125
Amendment of lines	55	Compilation (of overlay program)	45
Array elements and trace level 2	37	Compiler	
Array header	60	file	110
Arrays in mixed language programming	60-63	input	25-31
ATLAS card code	10, 25, 54, 105, 132	input media	25
*ATLAS directive, the	54	listing	19
		output	21-24
BACKSPACE statement, the	11, 13	Compiling	
Basic peripherals		and running on different size processors	47
compiling from	25-26	from basic peripherals	25-26
object program	10	from disc	26-27
Batch Development System, the	87-104, 112, 121	from magnetic tape	27-29
data handling	88	mixed PLAN/FORTRAN programs	71
examples of use	101-104	Compiling system statements	xvii, 1-4
general description	87	format of	2
macro specifications	91-100	on disc	27
operation of	87	on magnetic tape	28
parameters	89	Composite file	5, 7
programming information	88	COMPRESS statements, the	
restrictions	89	INTEGER AND LOGICAL	48
standard default program description	89	DOUBLE PRECISION	49
tracing	88	Console typewriter	xvii, 16
Batch directives	52	Consolidation error messages	125
Batch terminator, the	52, 53	Consolidation, separate	23
Blank common areas	65	Consolidator	xvii
Block data segments (equivalent in PLAN)	65	listing	20
Block size	14	COPY subroutine, the	17
Bucket	6	Core store, character transfers in	17
end of file	137, 138	*CPCOPY directive, the	54
format	137	CREATE statement, the	8, 9, 11, 13, 14, 15, 16
Buffer size	12		
		Data blocks	139
Calling FORTRAN segments from PLAN	67-70	formatted (on magnetic tape)	139
Calling PLAN segments from FORTRAN	57-67	unformatted (on magnetic tape)	139
Calling sequence	57	Data bucket	138
Card codes		Data formats	
1900	132	on disc files	137-138
ATLAS	132	on magnetic tape files	139
Channel 0	16	DEFBUF subroutine, the	17
Channel description statements	9	*DELETE directive, the	54
disc	11	Deletion of lines	55
magnetic tape	13	Depth of overlay	44
Channel numbers	9	DEPTH OF OVERLAY statement, the	44
Character code directives	54	Different size processors, compiling and running	
Character codes	131-132	on	47
Character transfers in core store	17	Directives	52-56

Disc		Executive	xvii
channel description statements	11	Extended data programs	47
compiling from	26-27	Extended FORTRAN	iii
compiling system statements on	27		
editing of source program on	27	FDUMP subroutine, the	15
error conditions	117	FEPiLOG subroutine, the	59
READ FROM statement for	26	example of	59
Disc file(s)	5-7, 10-12	File	
data formats on	137	allocation	110
effect of source program statements for	10	names	6, 7, 12
formatted	137	names at run time	14
obligatory	110	renaming	15
opening of	11	structure	139
optional	111	FILE subroutine, the	14, 15
selective	87	Files	
structure	109	composite	5, 7
unformatted	138	disc	5-7
Document names	74	magnetic tape	7-8
Double buffering	10	named	6, 7
Dump and restart facilities	15	scratch	6, 7
Dump file		simple	5, 7
printing of formatted	16	FINISH statement, the	2
specifying the	16	Formatted data blocks (on magnetic tape)	139
DUMP ON statement, the	1, 21, 22, 23	Formatted disc files	137
		printing of	16
Edit phase	87	Formatted magnetic tape files, printing of	16
Editing	51-56	Formatted records	12, 13
file	51	FORTBATCH macro, the	95
of source programs on disc	27	FORTBATCHFD macro, the	95
of source programs on magnetic tape	29	FORTRAN Batch Development System, the	87-104, 112, 121
Editor		FORTRAN macro, the	77
description of	51-52	FORTRAN segments	
input to	51	calling from PLAN	67-70
output from	51	incorporated in a PLAN program	72
Efficiency		FORTRANCOMP macro, the	79
considerations with E.D.S.	111	FORTRANCOMPX macro, the	81
run time (of overlaid programs)	45	FORTRANRUN macro, the	83
End of file		FORTRANSEMI macro, the	85
bucket	137, 138	FORTSEMICOMP	21, 22, 111
markers in object program input data	15	FOVER subroutine, the	69
END statement, the	2	FPROLOG subroutine, the	58
END FILE statement, the	11, 13	example of	59
Entry points	71	Free space subfile	5
Error action		Full listing	19
normal, with trace level 1	34		
with trace level 2	36	GENAH subroutine, the	61
Error conditions		example of	61
using disc files	117	Generation number	6, 7
other	117-131	GEORGE 1 and 2	73-107, 112, 121
Error detection example	39-41	introduction	73
Error detection levels, mixed	39	running #XFAE under GEORGE	121
Error diagnostics		use of GEORGE (in installation)	112
compilation	33	GETAH subroutine, the	60
object program	34-42	example of	61
Error messages	33		
consolidation	125	Hardware requirements	113
Error numbers			
compilation	123-125	ICLA-DEFAULT file, use of the	22, 111
execution	127-130	*IN directive, the	52
Error trap facility, the	35	Initial listing statements	19
example of	41-42	Initial statements	1
Errors, user defined	39		

Initialization of a system	109-111	Missing segments	33
Input		Mixed error detection levels	39
from simple files	29	Mixed language programming 1	57-72
from subfiles	27	Mixed language programs	47
to #XMED (the editor), examples of	55-56	Mixed PLAN/FORTRAN overlaid programs	69
INPUT statement, the	9, 11, 13	restrictions in using	70
Insertion of lines	55	Multi-level subfile structure	5
Integrity code	7		
Intersegment listing statements	19	Named files	6, 7
Intersegment statements	1, 2	Names	
Intervention, by operator	39	file	6, 7
		subfile	6, 7
Job	73	Nested subfile	5, 7
Job description	73	NEWBATCH macro, the	97
Job descriptions, user defined	105-107	NEWBATCHMT macro, the	99
Job directives	53	*NOLIST directive, the	53
JOB command, the	73		
		Object program	109, 114
Label errors	33	error diagnostics	34-42
*LAST directive, the	54	optionally overlaid from magnetic tape	45
Level 00, level 1, level 2, etc. subfiles	5	output of	22
Library file	110	Obligatory disc files	110
Library subfile	29	OMIT statement, the	48
LIBRARY statement, the	30	Opening of disc files	11
use of	30	Opening of magnetic tape files	13
Line editing directive	55	Operating instructions	115-116
*LIST directive, the	54	Operator intervention	39
Listing	19-20	Operators' guide	113-121
compiler	19	Optional disc files	111
consolidator	20	*OUT directive, the	53
full	19	Output of object program	22
short	19	OUTPUT statement, the	9, 11, 13, 15
Listing statements	19	OVERFL subroutine, the	35
initial	19	Overflow	34
intersegment	19	Overlaid programs	
*LOAD directive, the	53	mixed PLAN/FORTRAN	69
Logical unit number	9	restrictions in using	70
		Overlay	
Macro AMENDBATCH, the	91	area	43
Macro AMENDBATCHMT, the	93	system	43
Macro FORTBATCH, the	95	units	43
Macro FORTBATCHFD, the	95	Overlay program, example of	45-46
Macro FORTRAN, the	77	Overlay programs, program description	
Macro FORTRANCOMP, the	79	statements for	43
Macro FORTRANCOMPX, the	81	OVERLAY statement, the	43
Macro FORTRANRUN, the	83	OVERLAY PROGRAM statement, the	1, 44
Macro FORTRANSEMI, the	85	OVERLAY SEGMENTS statement, the	1, 44
Macro NEWBATCH, the	97	Overlays	43-46
Macro NEWBATCHMT, the	99	standard functions in	45
Magnetic tape			
channel description statements	13	Paper tape code	131
compiling from	27-29	Parameters common to standard GEORGE	
compiling system statements on	28	1 and 2 macros	74
editing of source programs on	29	Peripheral transfers, suppression of	18
object program optionally overlaid from	45	Peripheral usage (in mixed language	
Magnetic tape files	7-8, 12-14	programming)	71
data formats on	139	Peripherals, basic	10
effect of source program statements for	12	use of	113-114
opening of	13	Permanent area	43
printing of formatted	16	PLAN segments, calling from FORTRAN	57-67
MAP statement, the	20	PLAN segments incorporated in a FORTRAN	
Master segments	71	program	71

Printing of formatted disc and magnetic tape files	16	multi-level structure	5
Priority	48, 114	nested structure	5, 7
Process phase	87	semi-compiled	29
Program description statements	1	simple	5, 7
for overlay programs	43	single level structure	5
Program dump file	111	top level	5
Program structure	1-4	Subprogramming	71
Program testing	18, 33-42	Suppression	
PROGRAM statement, the	1	of peripheral transfers	18
Programs, mixed language	47	of trace output	39
Purge date	6	*SWITCH directive, the	54
		Syntax checks	22
		System description	109-112
READ statement, the	11, 12, 14	System initialization	109-111
READ FROM statement, the	2, 25, 26, 27, 28, 29	System input file	88
Record format	12, 14, 137	System name	9
RENAME subroutine, the	15		
Renaming, file	15	Terminator statement	2
Retention period	6, 8	Testing, of programs	18
REWIND statement, the	11, 13	Top level subfiles	5
Run time, file names at	14	*TPCOPY directive, the	54
Run time efficiency (of overlaid programs)	45	Trace level 0	39
RUN statement, the	23	Trace level 1	34
		Trace level 2	36
Scratch files	6, 7	Trace output channel	34
SEGMENTS statement, the	1	Trace output, suppression of	39
SEGMENTS MISSING message, the	33	Trace steering list(s)	37
Selective disc file	87	format of	38
Semi-compiled	xvii	Trace triggering statement	38
Semi-compiled output file	111	Tracing	34
Semi-compiled subfile	29	every statement	38
SEMICOMPILED statement, the	29	Transfer of control in mixed language	
use of	30	programming	57
SEND TO statement, the	21, 25, 111	Triggering statement	38
Separate consolidation	23		
Short listing	19	Unformatted data blocks (on magnetic tape)	139
Simple files	5	Unformatted disc files	138
input from	29	Unformatted records	12, 13
Simple subfiles	5, 7	Unit number, logical	9
Single level subfile structure	5	USE statement, the	9, 11, 13, 15, 16, 17
Source program		User defined errors	39
on disc, editing of	27	User job descriptions	105-107
on magnetic tape, editing of	29		
Source program statements		*WARN directive, the	54
effect of, for disc files	10	Warning character	54
effect of, for magnetic tape files	12	Word 30, use of	71
Standard functions in overlays	45	Work assembler	112
Standard (GEORGE 1 and 2) macros	74-86	Work file	111
macro specifications	77-86	WORK statement, the	6, 22
parameters common to	74	Write permit ring	8
programming information	74	WRITE statement, the	11, 12, 14, 16
STATEMENT TRACE output	36		
Steering lists, trace 2 with	37		
Subfile directive, the	52, 53		
Subfile names	6, 7		
Subfile terminator, the	52, 53		
Subfiles			
free space	5		
input from	27		
level 0	5		
level 1, level 2, etc.	5		
library	29		

