The FORTRAN system for Orion

By R. Taylor and D. A. Harragan*

The Orion FORTRAN system allows the compilation, assembly, and editing of programs written in FORTRAN or a symbolic assembly language. As it is not the first FORTRAN system ever written, most of the programming for it was done in FORTRAN. Great attention is paid to the needs of non-specialist programmers.

1. Introduction

The Rutherford High Energy Laboratory of the National Institute for Research in Nuclear Science exists to meet the needs of nuclear and high-energy physicists, many of whom are visitors from other similarly equipped laboratories. Their needs therefore include a computing installation which provides a powerful and easily accessible FORTRAN system offering compilation of FORTRAN subprograms, assembly of precompiled and newly compiled subprograms together with necessary library routines, correction of programs, and extensive debugging facilities at execution time. The FORTRAN Master Program (FORMAP) has been designed to supply these features, and is now running on the Rutherford Laboratory ORION computer, which has a 16K core store, a 32K-word drum, 7 magnetic-tape decks, three paper-tape readers, two paper-tape punches, and a line printer but no punched-card equipment. FORTRAN uses about 12K of core, 22K of drums, 5 decks, a reader and a punch, and the printer. Care has been taken, both in formulating the source language and in writing the compiler which implements it, to offer guards against the type of mistake, at compilation time and execution time, which seem commonly to beset FORTRAN users, many of whom are not professional programmers. Extensive use has been made of OMP, the Orion Monitor Program which is supplied as software with the machine.

2. Alleviation of programmers' difficulties

(*a*) *At compilation time*

Our observation shows that FORTRAN programmers have trouble with the following points:

(i) Storage Allocation. To help here, we have reformed the EQUIVALENCE statement so that only one identifier (at most) in an equivalence set may be declared COMMON. If there is one, the other members of the set also go into COMMON. EQUIVALENCE statements never re-order COMMON, the layout being governed solely by the order in which identifiers occur in COMMON statements, and by the associated DIMENSION statements.

(ii) Mixed-mode arithmetic expressions. We allow these.

(iii) Input and output statements, and FORMAT. Some FORMAT checking is done at compilation, to reduce the chances of the library input/output routines having to report trouble at execution.

(iv) Understanding the error diagnostic messages produced by the compiler. The first pass of the compiler always prints the source program together with a reference number for each statement. On discovering an error, it prints a message immediately beneath the line in which it was found, pointing when possible to the column containing the offending character. Error messages printed during the second pass give the reference number of the faulty statement, and the third pass, in which storage assignment is worked out, mentions by name identifiers whose allocations are ill-defined. An Error Book explains in detail the meanings of the printed error messages.

(*b*) *At execution time*

We decided that such events as attempts to evaluate square roots or logarithms of negative numbers, to mishandle peripheral equipment, to continue computation after floating-point overflow has occurred, should not be allowed to pass undetected. Instead, all library routines have been written so that the presence of illegal operands calls in a special utility routine ERRORTRACE which is always assembled into the program. This routine prints on the control desk monitoring Flexowriter a message containing the job-name, the name of the offended library routine, the nature of the error, and a back-track list of routine names showing the flow of control by which the library routine was entered. ERRORTRACE is also entered when a data tape contains punching which conflicts with FORMAT or contains illegal characters. If the programmer has made an AFTER ERROR statement (see below, Section 3.5) ERRORTRACE then passes control to the nominated routine. The default action is to call the library routine EXIT, which, however, does not abolish the program but halts it in a re-runnable condition, and so informs the operator.

More detailed debug printing can also be obtained. Each subprogram is compiled with a routine entry trap which may be activated by the operator depressing a key on the control desk. (This has obvious difficulties for a time-sharing machine like ORION, but the difficulties are for the operating staff, not the programmers.) When the trap is switched on, the program halts on the next subprogram call, asking the operator to nominate a sub-

* *Rutherford High Energy Laboratory, Chilton, near Didcot, Berkshire.*

program. It then prints the names of all subprograms which it enters, until the nominated one is met, when it halts again. The operator can then ask for OMP monitoring, e.g. printing out successful jumps and the contents of specified regions of core or drum stores. Thus the dynamical effects of a suspect subprogram can be found.

3. The FORTRAN Source (Orion) Language FORSOL

In specifying this language the following facts, most of which conflict with each other, had to be taken into account:

(*a*) FORSOL should not depart from the widely known FORTRAN II.
(*b*) Many Orion programs may ultimately be run on Atlas.
(*c*) FORTRAN IV differs in some details from FORTRAN II and may become equally widely accepted.
(*d*) The Orion hardware imposes certain restrictions.
(*e*) A working system had to be ready when the machine was installed.
(*f*) The language had to be describable to non-specialist programmers.

Points (*a*), (*b*), and (*c*) involve a consideration of the dialects of FORTRAN, a subject well covered (except for Orion) by I. C. Pyle in *Comm. of the A.C.M.*, Vol. 6, No. 8, p. 462, August 1963.

Points arising from (*d*) were easily settled. SENSE-LIGHT and SENSESWITCH are illegal. IF ACCUMULATOR OVERFLOW, IF QUOTIENT OVERFLOW, and IF DIVIDE CHECK were all made equivalent to a new form IF OVERFLOW.

The main consequence of (*e*) was that Arithmetic Statement Functions were dropped. Instead programmers have to write function subprograms, which in any case are more general and if wrong can be individually recompiled.

The above disposes of the ways in which FORTRAN II programs might fail on Orion. The remaining features of the language were fixed only after considerable thought. Usually we gave greatest weight, when considering some proposed incompatibility, to the intentions of our colleagues at A.E.R.E., Harwell, who were at that time specifying the dialect of FORTRAN to be used on Atlas. Fragmentation of a symbolic programming language into dialects is a serious evil for programmers, but we believe that the FORSOL dialect will allow many FORTRAN II programs to run unchanged on Orion, and many FORSOL programs to run unchanged on Atlas. The incompatibilities are intended to be of use in programs which will only run on Orion, e.g. programs which handle massive data streams coming from the Direct Data Connection by which Orion is attached to nuclear physics measuring equipment. Thus there are several open standard functions for performing shift operations, which on another computer would simply have to be supplied as function subprograms of the same name. One can also incorporate machine-language instructions, but this of course destroys all possibility of compatibility.

A full specification of FORSOL cannot be given here, but the following survey may indicate some of the ways in which FORTRAN has been generalized.

3.1. FORSOL allows the use of integer expressions as control parameters, e.g. as DO limits and steps, and as the index of a computed GO TO.
3.2. Arrays can have arbitrary dimensionality, and each subscript can be an arbitrary integer expression.
3.3. Modes LOGICAL and TEXT have been introduced, which permit single-word variables to be set to arbitrary bit-patterns or Hollerith character-strings. LOGICAL variables may also be equated to arithmetic relations, in which case they assume one of the values .TRUE. or .FALSE.. Such variables (and relations) can be combined be the operators .AND., .OR., .ER., and .NOT. to form a logical expression.
3.4. An alternative (not compulsory) form of IF has been introduced:
IF (logical expression) n_1, n_2
which sends control to statement n_2 if the expression is .FALSE., otherwise to n_1.
3.5. AFTER ERROR SUBR defines SUBR as the restart routine after ERRORTRACE.

4. Efficiency of implementation

As usual with FORTRAN compilers, common subexpressions within one statement are evaluated once only. Division by floating-point constants (surprisingly common in FORTRAN programs) is converted to multiplication by the reciprocal constant. $X**2$ becomes $X*X$; other exponentiations invoke library functions. The common case of an IF statement having the next statement as one or two of its successors is dealt with economically.

Subscripts receive rather novel treatment. Subscripts involving non-linear expressions in variables are evaluated when met, and no attempt is made at simplification; repetitions within one statement are detected. It is the ideal of many FORTRAN compiler writers to reduce the treatment of linear subscripts to evaluation at points of definition only, together with incrementation within DO loops. We felt that after perhaps an extra man-year we could have produced such a compiler, which would be half as fast as our present one, although object programs would be more efficient. Furthermore, observation shows that many FORTRAN programmers use only the simplest possible form of variable subscript, as in $X(N + 3)$. Now our arithmetic compiling routines frequently want to compile an instruction in which one or both of the main addresses must be modified not by the contents of an index register but by the contents of the register indicated by the appropriate address of the preceding instruction. If this is simply an integer variable, it need never be loaded into an index register at all. Accordingly we recognize such modifiers and refrain from copying the variable to an index register until the context shows this to be necessary.

Even complicated linear subscript combinations are simplified as much as possible; thus X(2*N*3 + K*4, M −N) is recognized to mean the same, within one statement, as *X*(4*K + 6*N, −N + M).

5. The assembly languages FALAN and FAMAP, and the assembler

FALAN is not the language into which the compiler translates, but an assembly language by means of which the programmer can write subprograms using all the facilities of the Orion Machine Code. It allows cross-references between FORTRAN-compiled routines and other FALAN routines, and permits the use of symbolic identifiers, symbolic labels, and label-setting equations.

FAMAP is the compiler output language, consisting of subprograms in relocatable binary plus 3 relocation maps per subprogram, one for relocation within a subprogram, one for relocation between subprograms, and one for relocation of the entire program. The Assembler can work out all the relocation (supplying any needed library subprograms), determines core and drum requirements, and incorporates a Utility Region into each complete program. This region contains the routine ERROR-TRACE, the routine CHAIN (the chapter-changing sequence), an entry pseudo-chapter, and several entry-points by means of which a program can be re-entered, after a halt, at the beginning of the main-level program of any chapter. The input and output buffers for the compiled program are also laid out here. The Assembler output for each chapter may be to drum or magnetic tape, as directed by the programmer. A program which is not to be kept in absolute binary will also be automatically supplied with a short loader routine, written in the Orion Basic Input language.

6. The user's view of the system

It was felt that neither punched cards nor punched paper tape were the ideal medium for storing large multi-chain FORTRAN programs in which scattered subprograms occasionally need recompilation. Accordingly the FORTRAN Master Program was designed so that the only paper tapes the programmer need handle are those containing his subprograms in FORTRAN, or in FALAN if he needs machine coding for a particularly crucial operation. The compiled routines are held in FAMAP on a magnetic tape which belongs to a programmer, and on which he can keep the FAMAP versions of several programs. The Assembler can convert such a program into a relocatable binary program held on a public magnetic tape containing many such programs, or into relocatable binary or absolute binary on a private magnetic tape. These tapes never leave the machine room. The FORTRAN system printed output includes information telling the programmer the code-numbers of these tapes and what subprograms they incorporate.

Editing facilities are available whereby the programmer can direct that subprograms be deleted from his tape, or be replaced by new ones of the same name coming in via the FORTRAN of FALAN languages or both, or be supplemented by entirely new subprograms.

Load-and-go facilities have not yet been provided. It is thought that the FORTRAN system works most efficiently when processing a steady stream of jobs without interruption for execution, which in many cases would involve re-loading one or more tape decks, an operation whose time is measured in minutes because FORTRAN needs 5 of the 7 tape decks, of which one is usually regarded as being unavailable owing to maintenance requirements. To obtain execution, the programmer simply has to supply a short length of paper tape defining his job name, his peripheral requirements, and estimated running time.

7. Writing FORTRAN systems in FORTRAN

The great and often unappreciated power of FORTRAN as a data-processing language is illustrated by the fact that the greater part of the Orion FORTRAN System was actually written in FORTRAN. The first version of the FORTRAN compiler was written in FORTRAN II together with a few routines in FAP. It was executed on an IBM 7090 and produced an object program on punched cards which were converted to paper tape and read into Orion. This FORTRAN II version accepted FORSOL as source language. Next, a new version of the FORTRAN compiler was written in FORSOL and compiled by version 1 on the 7090 so that then we had a version which would work inside Orion, and which could be used to generate improved versions of itself. This process is still continuing, the recently completed editing feature greatly facilitating the whole procedure. At present the compiler puts out about 350-400 machine-language instructions per minute. The Assembler is so fast that time estimates are difficult.

8. Acknowledgements

We wish to acknowledge the great assistance and co-operation we have had from Mr. C. S. L. Atkinson (who planned most of pass 2 and pass 3 of the original compiler) and Mr. R. Wilson, who were both at that time with Ferranti Ltd.; from Dr. I. C. Pyle, Mr. E. B. Fossey, Miss B. Stokoe, Mr. P. Bryant, and Mr. R. MacLatchie of A.E.R.E., Harwell; and from our colleagues at the Rutherford Laboratory, Mr. J. W. Gardner, Mr. P. A. Denny, and Mr. D. J. Wagon.

Reference

—(October, 1963). *A Primer of Fortran Programming for use on Atlas and Orion Computers* (based on work done by teams at AERE, Harwell, and Rutherford Laboratory, Chilton), published by I.C.T. Ltd., 68 Newman Street, London, W.C.2, Reference list C.S. 390.