it cannot, however, provide quantitative information on the projected system in the same way as a simulation package is able to do. It can, however, provide a completely accurate and relevant picture of an existing system and quickly isolate inefficient areas, if they exist.

---

A complete and intensively used computing service has been moved from a second to a third generation computer with a minimum of disturbance to user programming practices. The software techniques are designed to facilitate similar transfers to future machines and to avoid the waste of discarding strongly machine-dependent software. The critical features of the software design are discussed in relation to the systems programming effort required and objectives achieved.

# Bridging the generation gap

J.K. YARWOOD and G.E. MILLARD
*Edinburgh Regional Computing Centre*

## Introduction

Since the inception of the Regional Centre in 1966 to serve Edinburgh University and Edinburgh Research Councils, a substantial computing work load has been continuously supported on a variety of batch processing systems. At the same time an objective has been sustained to progress towards the introduction of a fully interactive system for all or part of the service. These dual aspects of the Regional Centre's existence have caused considerable attention to be directed to the problems of mobility and complementary compatibility. The work described in this paper has resulted from one principal requirement: to provide a high degree of continuity for the users despite changes in the operating system base.

In the 1960's service to Edinburgh users commenced on the Manchester Atlas by paper tape transmission link. This was augmented by the use of off-site English Electric KDF9's with an Edinburgh-created Atlas Autocode system developed at the Glasgow and Newcastle installations. In 1966 the Regional Centre service for University and Research Councils started, based on a local KDF9 as an interim machine pending delivery of an ICL System 4-75. The service continued using the Chilton Atlas and subsequently was augmented with the NEL Univac 1108 in 1967 and the Newcastle 360/67 in 1968. Since then the basis of the computing service has been an IBM 360/50 (now replaced by a 370/155) running under OS (with 10 remote job entry stations) and an ICL System 4-75 using the manufacturer's J operating system (25 teletypes, [1]) for two years, but now running almost entirely under the Edinburgh Multi-access System (100 teletypes, EMAS, [2]). OS and J are non-paged batch processing systems, while EMAS provides large paged virtual memories for fully interactive or batch use.

Continuity for users during the far-reaching changes of the past six years has been achieved in the following important areas:

1. Compilers for IMP and FORTRAN. The former is a local derivative [3] of Atlas Autocode [4] and is much used in Edinburgh, especially for systems software. The latter, as the most used language for the international exchange of working programs, is supported to the level of the IBM FORTRAN G compiler as the de facto working standard pending publication of the new ANSI standard in 1972/73.
2. Data bases. Programs were written for the initial transfer of users' data into the third generation operating systems. Equivalent or identical access methods now exist on these operating systems in Edinburgh allowing users' programs and programming techniques to be freely transferable between the systems, while utilities are available for data transfer via magnetic tape. File facilities of both languages include the sequential and direct access methods familiar to IBM FORTRAN users. IMP has in addition the feature of character-by-character I/O to and from sequential files.
3. Job control. For an installation having 4,000 users and running 10,000 jobs per week, the problems and costs of teaching, documentation and advisory support must be obvious. They are minimised by having a standard job control language for all batch jobs up to a defined level of complexity (satisfying practically all users) for all of the alternative operating systems.

However, benefits of the software techniques developed to provide continuity to users are not limited to the satisfaction of users and the minimisation of advisory support. It is estimated

TABLE

| | ICL System 4 and IBM System/360 | | | Mobility: proportions of investment movable to: | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1. New operating system on equivalent hardware | | 2. New hardware | |
| | Source language | Size, Kbytes, each system | Investment, man-years, all systems | Program | Technique | Program | Technique |
| Compilers: | | | | | | | |
| (i) FORTRAN | IMP | 80 | 4.0 | 100% | 100% | 70% | 70% |
| (ii) IMP | IMP | 85 | 4.0 | 100% | 100% | 70% | 70% |
| Execution environment: | | | | | | | |
| A (see note) | IMP and | 60-70 | 3.5 | 20% | 50% | NIL | 50% |
| B (see note) | Assembler | 20-30 | 3.0 | 20% | 50% | NIL | 50% |
| System library (fully sharable): | | | | | | | |
| (i) FORTRAN support | FORTRAN | | | 100% | 100% | 100% | 100% |
| (ii) IMP support | IMP and | 77 | 3.0 | 90% | 90% | 70% | 70% |
| (iii) Mathematical functions | Assembler | | | 90% | 90% | 70% | 70% (see note C) |

Notes:
A. Program management facilities: editor, linker, librarian, job control language analyser, execution/batch control, library loading.
B. System Interface Module, loader, object output routine, basic machine interface, device drivers.
C. The shortfall, 90%-70%, is intended to represent the proportion of IMP support and Mathematical functions coded in Assembler.

(see Table) that the move to the third-generation operating systems during 1966-1971 has involved 17½ man-years of systems programming effort. Of this, half has been spent in developing and refining compiling techniques for the Centre's two main languages, FORTRAN and IMP. Experience with a series of current computers (including non-byte-oriented machines) indicates that perhaps 70% of this effort is effectively preserved for the future in the form of system-independent high-level-language programs. Similarly, large proportions of the system library material for the two languages, including standard routines and functions and I/O support, would be directly applicable to a range of new hardware.

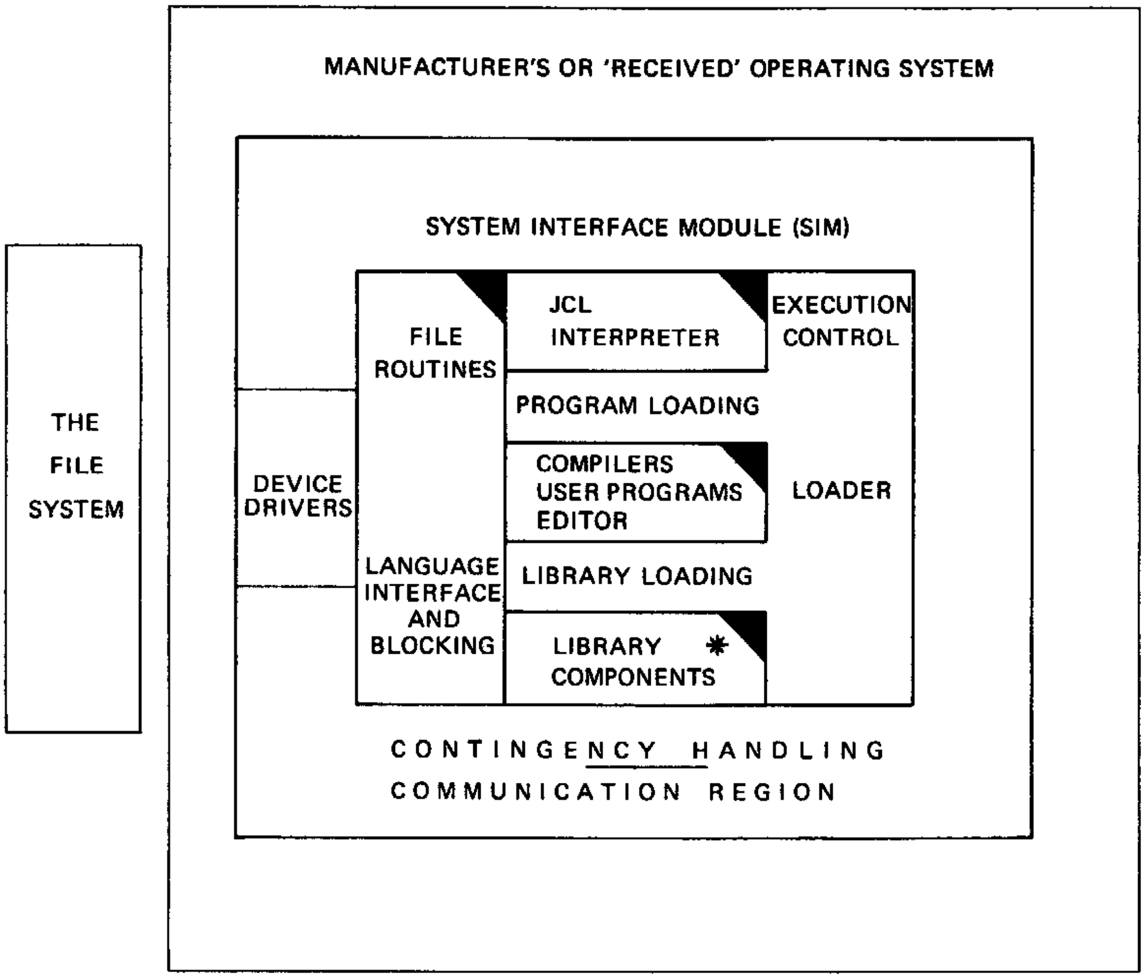
The key to this economy of effort (both in developing the third-generation user service, and in providing for the future) lies in the adoption of a high-level language for implementation. The IMP compiler is thus undoubtedly the heart of the Regional Centre scene. Apart from its extensive use as a user programming language, IMP is the source language of the FORTRAN and IMP compilers, a very large proportion of each 'subsystem' execution environment and library material, and the whole of the EMAS multi-access supervisor and subsystem. The FORTRAN compiler, on the other hand, was initiated in 1966 as being the only practical solution to the requirement for a shareable compiler for the proposed multi-access system, but has provided incidentally the substantial benefits of excellent diagnostics and run-time efficiency, mixed-language capability and above all mobility.

## Mobile components

The viability of techniques for inter-machine transfer both of individual software components and of an integrated subsystem is measured by the ratio of investment in new programming involved in the transfer to that which would be required to re-create the system from scratch on the target machine. The following are critical areas of major investment:

Compilers.
Program execution environment.
System and user libraries.
The file systems.

For a computing system implemented using the assembly code of the machine, trauma and dis-



◥ truly mobile and written in IMP.

✱ except those coded in assembly language

ruption can confidently be predicted in each of these areas, both for the users and for the systems programming staff, when the time comes for the machine to be replaced by one of a different type. Considering each area in turn, we indicate, in conjunction with the Table, the extent to which waste is inevitable when the current machines are abandoned and the functions which have been isolated into machine-independent programs in the Edinburgh systems. The columns labelled 'Technique' under 'Mobility' in the Table are intended to represent a purely subjective assessment of the extent to which the experience gained with current systems might be directly applied to new systems. On the other hand the columns labelled 'Program' under 'Mobility' are intended to indicate the actual proportions of current software which could be transferred to a new system once compilers have been provided.

Subsequently we discuss the software and data-handling interfaces in the Edinburgh systems giving lists where appropriate of the functions which must be provided in the new environment to enable the replication of the existing computing system. The Figure is an attempt at pictorial representation of the interfaces between components, showing how a System Interface Module (SIM) comprising a selection of fundamental functions is used to isolate the central investments from their surroundings so that their development in a system and machine independent manner has been encouraged.

Questions of efficiency of a system implementation using these techniques are ultimately dependent on target hardware. Notable effectiveness has been achieved in several operating systems on admittedly similar hardware, and indeed efficiency has been an important consideration in establishing the software interfaces. The critical fact is that after a new implementation has been effected at possibly low efficiency, a year of tuning the basic components, in particular the compilers, can be made extremely productive and improvements can be continuously passed on to users, hopefully quite without any step change in programming practices and usually without the users' immediate awareness. That this is possible has already been adequately demonstrated in the Edinburgh environment, both on the main service computers and on a variety of other machines, as is mentioned below.

### Compilers

The machine-independent aspects of high-level-language compilation have been widely discussed. Suffice it to say that in Edinburgh the FORTRAN and IMP compilers have been developed with the following prime objectives:

Rapid single-pass compilation.
Excellence of compile-time and run-time diagnostics.
High run-time efficiency for debugged programs.
Sharability of compiler and object code.
Mobility.

Both compilers are written in IMP. Although they have in-line machine-code in areas where execution speed can be significantly enhanced, versions are maintained which are IMP throughout. While their single-pass, non-overlaid structure is relatively extravagant of core, speed is enhanced and the objective of mobility is served by reducing their dependence on sophisticated I/O requirements, and their size is by no means embarrassing in machines having upwards of 100 Kbytes of core.

The object code generation routines are well-defined, allowing future adaptation for the production of object programs for different machines. Further, a compiler exists which compiles a generous subset of the full user-oriented IMP into an assembly language for a hypothetical single-accumulator, three-index-register machine. This compiler will compile itself, and the full IMP compiler will compile both itself and of course the FORTRAN compiler. Bootstrapping procedures are thus available on a broad front for mobility to a variety of environments. IMP compilers have been produced (not necessarily by Regional Centre staff) for:

IBM System/360 (OS)
ICL System 4 (J operating System)
ICL System 4-75 (EMAS)
IBM System/360 (Michigan Terminal System – MTS)
ICL System 4 (Multijob)
Univac 1108
CTL Modular One
DEC PDP8
DEC PDP9
DEC PDP11
DEC PDP15

Full user facilities are maintained only on the first three of these. For the others, at the present time, the compilers are in use primarily for systems programming and communications work, or for teaching.

On the System 4 and System/360, the object code format is different from that specified by the manufacturer, though converter programs are available for the creation of modules loadable by the manufacturers' software. The reasons for the decision to proceed independently are explained below, but one consequence is that object code is freely movable between the first five operating systems in the above list, giving considerable advantages in the maintenance of compilers and of system and user libraries. The form of object modules on an external medium is a sequential file of fixed-length blocks.

### Program execution environment

Job Control Language (JCL) has hitherto been mainly the province of the operating system writers, usually the manufacturers. Frequently its status as unfortunate necessity in the process of describing a job and having it initiated is only too evident. Clearly some apology must be made for the decision at Edinburgh to adopt the notation of IBM's OS/360 JCL for implementation on other and subsequent systems. When the Regional Centre's hired 360/50 had been in use for some twelve months, and was carrying essentially the whole of the batch computing service, the problem first became plain: the third-generation operating system needs a great deal of information for job scheduling, and a correspondingly large teaching and advisory effort is involved in establishing users on the machine. Since the JCL does provide a description of the primitive functions of batch computing, with a description of file usage of sufficient generality, the replication of the OS/360 JCL in the System 4 operating systems has enabled the user to utilise the same hard-won expertise in batch job description equally on the three main systems.

The Job Control Language interpreter now running on the System 4 is not entirely operating-system-independent, but the canonising of the input text is the main task. Tables of information are created to suit the operating system. The program is written in IMP and will readily execute on any machine for which an IMP compiler is provided. Indeed for small FORTRAN and IMP jobs (particularly for teaching purposes) requiring strictly limited CPU time and file usage, a version of the interpreter running on the 360 itself enables generous savings in OS/360 overheads by reading and, with other components, controlling the execution of, a stream of batched FORTRAN or IMP jobs as a single OS/360 job. The investment in this program of 20 Kbytes is not large (about $\frac{1}{2}$ man-year) but it will survive for the foreseeable future, and its users will require no further re-education.

Turning now to the functions of core allocation, compiler, program and library loading, basic I/O, the handling of interrupts and other contingencies, and file accessing – all these are necessarily quite strongly related to the hardware and to the supervisor. Nevertheless, in each case the central functions have been identified and suitable interfaces established, as described below, which clarify the issues involved and provide the framework round which a future batch system would be built. Perfecting the execution of these functions on the three systems has absorbed in total about $6\frac{1}{2}$ man-years, for 80 to 100 Kbytes of software, depending on the system. Perhaps 20% of this is written in assembly language, the rest in IMP. The whole would be largely inapplicable, except in concept, to new hardware.

## System and user libraries

Equivalent library facilities have been provided for batch users on the J, EMAS and OS systems. These allow:

1. The compilation of separate (external) routines or files of routines in FORTRAN or IMP.
2. The linking of external routines, with mixed-language capability (the object modules having identical format).
3. The storage of external routines in a user library.
4. A full complement of mathematical routines in the system library.
5. Automatic loading of routines, to satisfy external references, from user and/or system libraries.

The means of implementing these facilities has necessarily been different for each system. The routines amount to some 77 Kbytes of code, mainly in FORTRAN and IMP, with some in assembler, the investment amounting in total to about 3 man-years. Each item is of course sharable, and in the EMAS system this is fully exploited. Most of the investment could be transferred with ease to further systems.

There is an increasing demand from research workers for existing large packages, such as SPSS and BMD, to be made available. They are usually written in FORTRAN and the majority originate from universities in the United States. Moving such packages inevitably involves some modification where system-dependence has been built in, but again there is benefit from the continuity of the user interface in the Edinburgh systems. The service is now backed with a substantial range of such powerful packages.

## The file systems

In spite of the diversity of file system concepts between the three systems at the Regional Centre, much has been achieved in establishing a set of facilities which satisfy most needs of a general computing service. At Edinburgh there are few exceptional users requiring files in excess of two megabytes, or requiring rapid access to such large files. On the other hand, the 360 holds files totalling 400 Mbytes on behalf of over 50 university departments and other organisations. Similarly after six months of user service EMAS holds 2,000 permanent files, totalling 50 Mbytes, on behalf of 120 individuals, and these figures are growing steadily.

It is clear that files of the following logical types provide an acceptable range of facilities for most batch and interactive users:

1. Sequential files of text, considered as streams of characters or as records or lines of text.
2. Sequential files of 'binary' information, that is, copied bit by bit from its internal form or from an external medium.
3. Direct access files of similar 'binary' information in records of fixed length.

Using these file-types as a basis, users can if necessary simulate systems of greater sophistication, yet these facilities can be described with reasonable clarity in the job control language, and conform to the file types implied by I/O statements in programming languages commonly in use, particularly FORTRAN.

Taking the above list of file-types as part of a specification of the user/file-system interface, the following functions need to be replicated in transferring the system from one machine to another:

1. Device-driving for the physical I/O of blocks of data.
2. Interpretations between these blocks and the records to which a user program logically refers.
3. Transfer of the data, or of its description, between the user's program area and the routines performing the interpretations.

It is clear that the first of these is strongly machine or system dependent (indeed in EMAS the supervisor shields both user and subsystem from the external world through the virtual memory concept) and the last is language-dependent. The most convenient ways of handling data I/O on each of the three systems are significantly different. On OS the basic access methods are used (sequential and direct access); on J the 'system file format' is used for disk files and basic access (physical handling) for all other devices. On EMAS a completely different concept is involved: files are 'connected' or 'mapped' into a user's virtual memory and can be accessed without any explicitly organised I/O request.

However, in each case, the system-independent interface is provided at the record level, with the basic operations of select, get record, put record and position (for back-space, rewind and direct access). The mapping between data set numbers and actual files is a function of the system-dependent area and all tables maintained in support of this are private. By arranging that the system-dependent code provides the buffer areas, full advantage is taken of the special case of EMAS where those buffer areas are part of the actual file. The IMP character I/O facilities are regarded as a language feature and are provided by library routines accessing data through the 'record' interface.

Except for parts of the device-driving code, all functions have been coded in IMP and are machine-independent. Programming investment for the two languages amounts broadly to two man-years.

### Interfaces: program execution environment

In designing a bridging subsystem, the important decisions are: first, what are the interfaces between the subsystem and its supporting supervisors? Then, at what level should the division between system-independent and system-dependent software be pitched? We do not attempt to offer a best solution, but indicate the choices that have been made at Edinburgh.

We list the facilities available by high-level-language routine call at the interface between the System Interface Module (SIM) and the central and machine-independent programs, as illustrated in the Figure:

1. From device drivers for fast and slow devices: select channel, get a block or record, put a block or record, close channel. For fast devices the equivalent services with logical records can be obtained by suitable calls on the file routines, and this is the route by which most of the remaining software accesses the file system. The level at which device control is exercised depends strongly upon the environment. If the 'received operating system' provides facilities closely corresponding to those of the SIM interface to IMP programs they will obviously be used. Otherwise the appropriate code must be written as part of SIM to the extent required or allowed by the 'received operating system'.
2. Get date, time of day, send message to operator, set or read CPU timer.
3. Give the address of a Communication Region, which holds addresses of: lists of routine entry point definitions and external references (used in particular by the loader), system and user file descriptors, translation tables, free space limit references. The extent to which the Communication Region is used depends on the implementation.
4. A mechanism for the manipulation of program environment descriptions. In particular a stack or register/program-counter words is maintained, enabling control to be passed to chosen places on the occurrence of one of a set of contingencies. In this way the source language diagnostic routines are entered in the event of program error, whence control is passed to the next task for the subsystem. The errors or conditions may be asynchronous, and may be hardware or software detected. In detail, lists of typical functions and events encompassed by the mechanism are as follows:

<u>Contingency-handling services of the System Interface Module</u>

i Stack an environment description (up to 4 levels).
ii Unstack one environment description.
iii Unstack all environment descriptions.
iv Give number of stored environment descriptions.
v Signal an error with given identifier at the current level.
vi Signal an error with given identifier at the outer level.
vii Repeat the latest contingency report at the current level.

<u>Sample list of contingencies handled by the System Interface Module contingency mechanism</u>

i Illegal operation code.
ii Address error.
iii Fixed and floating point overflows, exponent underflow and divide error.
iv Operator interrupt.
v CPU time limit exceeded.
vi Printer output limit exceeded.
vii 'End of file' encountered.

### Interfaces: object program standards

The most important constraint required for EMAS (and desirable for other systems) is that code (sharable) and data (non-sharable) areas of programs should be separated. Compiler object output is therefore via an interface routine to which input is fragmented, with each record identified as code, symbol tables (for diagnostic purposes), non-sharable initialised data or external linkage data. This routine stores the object output (probably by suitable calls on System Interface Module facilities) in the form most convenient in the particular operating system.

Routines of FORTRAN and IMP use common stack, parameter-passing and linkage mechanisms. The latter provides static and dynamic linking capability [5] and allows the exploitation of the sharability of the code. Thus mixed-language working, including correct source language run-time diagnostics, is a standard feature of these systems.

### Acknowledgements

The following have been materially concerned in the development of the designs described here: Mrs M. M. Barritt, as Principal Consultant; P. D. Stephens, in control of the IMP language and compiler; H. Whitfield, D. J. Rees, A. S. Wight and S. T. Hayes in finalising and implementing the Edinburgh Multi-access System, which was initiated by the University of Edinburgh Computer Science Department in conjunction with International Computers Limited.

### References

1. YARWOOD, J. K., 'Access to an ICL System 4 computer by teletype and video terminals', Edinburgh Regional Computing Centre Pro-

gram Library Unit, Research and Development Notes, No. 2 (May 1971).

2. WHITFIELD, H. (Editor), 'System 4-75, Edinburgh Multi-access System Reference Manual', University of Edinburgh, Department of Computer Science (1971).
3. BARRITT, M. M., BURNS, J. G., McKENDRICK, A., and STEPHENS, P. D., 'The Edinburgh IMP Language Manual', Edinburgh Regional Computing Centre (1st edition, 1970).
4. BROOKER, R. A., and ROHL, J. S., 'Atlas Autocode reference manual', University of Manchester Computer Science Department (1965).
5. MILLARD, G. E., REES, D. J., and WHITFIELD, H., 'System 4-75, Edinburgh Multi-access System: Subsystem reference manual', University of Edinburgh, Department of Computer Science and Regional Computing Centre (1972).
6. MILLARD, G. E., FINCH, A. S., MARR, C. R., and AITKEN, W., 'The Edinburgh FORTRAN language manual', Edinburgh Regional Computing Centre (4th edition, 1971).
7. MILLARD, G. E., 'The Edinburgh FORTRAN Compiler and its environment', Proceedings SEAS XVI, Pisa, 1971, pp 318-27.
8. YARWOOD, J. K., 'Towards machine-independent processors', BCS Computer Bulletin, V14(7), pp 219-21 (July 1970).

## Discussion

Q. It appears that you adopted the JCL of OS/360 directly for the System 4. Did you consider developing your own JCL, and if not, why not?
A. The decision to hire a 360/50 to carry the service for 12 months or more pending evolution of the software for System 4 necessarily involved teaching and advisory support on at least basic OS JCL. Since it proved possible to replicate this JCL on the System 4 with very modest programming effort, this course was taken in order to minimise user re-training in JCL.

Q. You have constructed an interface and compilers to achieve portability but thrown away the possibility of using the manufacturers' software. Why?
A. This is not the case. Although object and load modules of the Edinburgh system are not directly compatible with those of the manufacturers, programs have been written to convert them to the manufacturers' (ICL and IBM) formats. Thus it is possible to interface to and to use provided software, and this is being done in certain areas. In addition, use of the manufacturers' software independently of the Edinburgh system is certainly not precluded, as the system described in this paper runs as an ordinary job in the machine, multiprogrammed with other jobs which may or may not themselves be instances of the Edinburgh system.

Q. If you do design your own JCL, won't you have difficulty in 'bridging the gap'?
A. There seems no reason to anticipate special difficulty with any particular form of a JCL, since a JCL text analyser can easily be made machine-independent.

Q. Is your system transportable to a CDC 7600?
A. The system has mainly been developed on byte-oriented machines which are fairly closely similar. Moving to further byte-machines would be comparatively straightforward, whereas going to other machines would require greater effort, mainly in adaptation of the IMP compiler for different word-length. Experience with IMP on certain non-byte machines indicates that this greater effort would be by no means excessive in view of the rewards to be gained.

Q. Does your system support ALGOL? If not, what are the problems?
A. It does not support ALGOL, mainly because IMP provides comparable and in many areas better facilities and the Edinburgh community is familiar with IMP. There would be no problems other than those already encountered and solved in providing an IMP compiler.

Q. Have you considered intermediate low-level language compiler output as a means of assisting compiler portability?
A. This is one of the techniques we have adopted. A compiler has been produced by Computer Science Department and Regional Centre staff which compiles an implementation subset of IMP into the assembly language of a hypothetical single-accumulator, three-index-register machine. This compiler has been used to bootstrap IMP to a Univac 1108, and is currently being used to provide IMP for DEC PDP-11's.