can be addressed in the same way as a simple variable. A short address length may preclude some array access optimisation, for instance if 'a' is a global array of fixed size a[200] could be accessed by a single instruction provided the address field was large enough. In fact the B5500 does not allow array accessing optimisation because the storage protection system depends upon access via the array word (descriptor). The optimisation produced by the ALCOR compilers (Grau, 1967), could be done on a machine with a short address length, but not the B5500.

Array bound checking is an area where special hardware can be used to great advantage. Unfortunately the hardware on the B5500 does not deal with the general value of the lower bound, so that explicit code must be generated by the compiler to subtract the value of this lower bound if it is non-zero. Options to do bound checking on other machines tend to be very expensive in processor time. The 1108, although having no built-in hardware for array accessing, has a convenient instruction for bound checking. With this instruction, a single test can be made to see if the operand lies within the range defined by two registers.

Apart from the production of compact code from ALGOL 60, it is clear that in many scientific fields non-conventional machines can have other substantial advantages. Array bound checking has already been mentioned, but other examples lie outside the scope of this paper, for instance distinction between data and program and the ability to share the available core store between processes. The majority of these advantages are in the field of operating system design, and so are not considered here. Such advantages are likely to have a substantial effect upon the performance of the compiling system itself, and the easy way in which such systems can be developed.

References

Burroughs B5500 Extended ALGOL Language Manual, 1966.

GRAU, A. A., HILL, U., and LANGMAACK, H. (1967). *Translation of ALGOL* 60, Berlin; Springer.

HAWKINS, E. N., and HUXTABLE, D. H. R. (1963). A multipass translation scheme for ALGOL 60, *Annual review in Automatic Programming*, Oxford; Pergamon Press.

HEINHOLD, J., and BAUER, F. L. (Editor). (1962). Fachbegriffe der Programmierungstechnik, Ansgearbeitet vom Fachausschutz Programmieren der Gesellschaft für Angewandte Mathematik und Mechanik (GAMM), 2 Anfl. Munchen, Oldenbourg-Verlag.

INGERMANN, P. Z. (1961). Thunks—A way of compiling procedure statements with some comments on procedure declarations. *CACM*, Vol. 4, No. 1, pp. 55-58.

KNUTH, D. E. (1964). Man or boy? Algol Bulletin No. 17, page 7, Mathematische Centrum, Amsterdam.

RANDELL, B., and RUSSELL, L. J. (1964). *ALGOL* 60 *Implementation*. APIC Studies in Data Processing No. 5, London; Academic Press.

SCOWEN, R. S. (1965). Quickersort, Algorithm 271. *CACM*, Vol. 8, No. 11, page 669.

WICHMANN, B. A. (1969). A comparison of ALGOL 60 execution speeds. National Physical Laboratory, CCU 3.

WICHMANN, B. A. (1970). Some statistics from ALGOL programs. National Physical Laboratory, CCU 11.

WICHMANN, B. A. (1971). The performance of some ALGOL systems. To appear in the proceedings of the IFIP congress 1971.

Correspondence

To the Editor
The Computer Journal

Sir,

Suggested Extension to FORTRAN IV

When endeavouring to translate an ALGOL program to FORTRAN recently, I came across a statement of the type:

for $I = 1$ **step** 1 **until** 10, 15, 20 **step** 10 **until** 100, $I + 100$
while $(B \wedge (A \leqslant C)$ **do**

A statement of this type obviously cannot be translated into FORTRAN without a great deal of complication.

On the other hand, it would seem a logical extension to FORTRAN to allow DO loops of an alternative type, of the general form as follows (or a similar form):

DO n I = /$n1$, $n2$/$n3$/$n4$, $n5$, $n6$/I + (integer variable or expression), (Boolean variable or expression)/

With this, the ALGOL expression above would become, in Extended FORTRAN:

DO n I = /1, 10/15/20, 100, 10/I + 100, (B. OR. (A.LE.C))/

The usual form of the DO loop would, of course, be retained. Parsing of the above would be distinguished firstly by the slash following the equals. The items between the slashes would constitute a 'DO' list element, similar to the for list element in ALGOL.

A further improvement might be to allow the use of negative and/or real stepping values in the suggested form, which would increase the power of FORTRAN considerably.

Yours faithfully,

A. J. FINN

'Aeschi'
Salthaugh Road
Keyingham
Hull HU12 9RT
11 October 1971