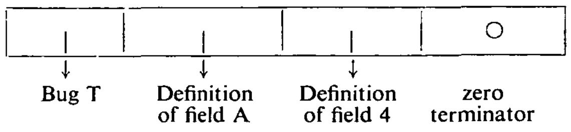
Appendix A note on BCL and the analysis of LSIX instructions

BCL is a general purpose programming language with special emphasis on data structures. Consider the sequence

```
FIELD IS (OSP., (EITHER 'T.', TIMEFIELD
                 OR     BUG, (EITHER FLDNAMES OR NIL.)
          OR  INTEGER ,'.', IF INTEGER LE 128, READFIELD)
          ,OSP., OCT := 0, PLANT )
```

which occurs in the main text of this report. The first two words indicate that this is a definition of the 'name' FIELD. That the rest of it is a parenthesised structure with commas indicates that FIELD denotes a structure of the type known as a 'group'. The commas between the 'objects' denote juxtaposition, and for alternatives the notation EITHER. . . . OR. . . . is used. The objects within a group may be literals or names. Character literals are enclosed with primes, numeric literals are obvious, also literal commands such as $x := z$, and literal groups (in parentheses). Names, which must of course be defined somewhere, but can be defined *passim*, may be names of variables, routines or groups. Group definitions may be recursive, i.e. the name of a group may appear in its own list of objects.

Suppose we encounter the object 'FIELD' when in the course of reading in, and the next characters in the input stream are TA4, a remote field. These characters are matched with objects in the group FIELD. The first object, OSP., is a built in group which recognises and skips over any number (including zero) of spaces. Next we have the first of three alternatives. The next two characters in the input stream are compared with the literal 'T.'. T is matched but period is not so this match fails and the second alternative is tried. The group BUG recognises T as the name of a bug or base-field and plants its address in the object area. The second object in this branch is itself a pair of alternatives, (EITHER FLDNAMES OR NIL.), which matches any number of field names and computes and plants the addresses of the corresponding field definitions. In this example, field names A and 4 are recognised and the corresponding addresses planted. Finally, after the successful matching of the second alternative, OSP. reads over any spaces, the variable OCT is assigned the value zero and the group PLANT plants the value of OCT in the object area. Thus as a side effect of the recognition of the remote field TA4 the following sequence of pointers is planted in the object area.



A second example is the special read-only field 64. (an integral power of two terminated by a period). As the first character is a digit, attempts to match 'T.' and BUG fail and the third alternative is tried. The object INTEGER is an integer variable to which the integer 64 is assigned. Then the period is matched and if the condition INTEGER LE 128 is satisfied the routine READFIELD tests that the input integer is an integral power of two and computes and plants the address of the field '64.'.

When BCL is used as a compiler compiler, commands written as objects in a group may generate and plant object coding as soon as source language instructions are matched. Alternatively the user may, if he so wishes, construct analysis records.

References

HENDRY, D. F. (1966). *A Provisional Manual for the BCL Language*, University of London Institute of Computer Science (Internal report).

KNOWLTON, K. C. (1965). A Fast Storage Allocator, *Communications Assoc. Comp. Mach.*, Vol. 8, pp. 623–625.

KNOWLTON, K. C. (1966). A Programmer's Description of L^6, *Communications Assoc. Comp. Mach.*, Vol. 9, pp. 616–625.

Book Review

Indices and Primitive Roots, by A. E. Western and J. C. P. Miller 1968; 385 pages. (London: C.U.P., £6 0s. 0d.)

This work incorporates and supersedes *Haupt-Exponents, Residue-Indices, Primitive Roots, and Standard Congruences*, published in 1922 by the late Lt-Col. Cunningham in collaboration with H. J. Woodall and T. G. Creak. It may also be regarded as a continuation of Jacobi's *Canon Arithmeticus*.

The editors denote by g, g', h respectively the least positive, the least negative, the least prime primitive root modulo P, P being an odd prime. It would be convenient to define also $G = g$ if $g \leqslant g'$, $G = -g'$ if $g' > g$. With this, the index of a (prime to P) given in the main tables is the least $n \geqslant 0$ such that $a \equiv G^n \pmod{P}$. The tables give (i) the complete factorisation of $P - 1$; (ii) g, g' and h; (iii) the indices of certain a; and (iv) the residue-indices ν = g.c.d. (ind a, $P - 1$).

Table 1 covers all P up to 50021, Table 2 the P between 50000 and 10^5 and $\equiv 1 \pmod{24}$. Table 3 goes up to 250000, with the stronger restriction $P \equiv 1$ or $49 \pmod{120}$; and in Table 4 $P \equiv 1 \pmod{120}$ and $P < 10^6$. This large range of P is made possible by restricting the range of a; in Table 1, a ranges over primes up to 37 and 6, 10, 12. With this information it is not too difficult to calculate the indices of other a, as explained in the introduction.

The original calculations were all done by hand or with a desk machine, and the method is explained in detail, with some subsidiary tables, so as to enable the reader to investigate primes P not given in the main tables. All the entries have, however, been checked at least twice, on the ACE computer at the National Physical Laboratory. As a result, the surviving editor (Dr. Miller) hopes that very few errors remain; the reviewer is unable to say whether he is right.

The tables should be very useful to workers in the field. They provide evidence for many plausible conjectures, e.g. that $g = g(P)$ defined above is of very low order of magnitude for large P.

G. L. WATSON (London)