Design and testing of the System 4 random number generator

By J. D. Beasley and Kathleen Wilson*

The design of the System 4 random number generator is described, and test results given. No particular originality is claimed, but a few conclusions by previous authors are commented on.

(Received March 1969)

1. Object

The purpose of this paper is to describe the tests used on the random number generator for the ICL System 4 series of computers, to describe the reasons underlying the choice of generator, and to present a summary of the results obtained. It is felt that these results may be of interest both to actual or potential System 4 computer users and to anybody wishing to produce a generator for use on some other machine with convenient facilities for 32-bit arithmetic; we do not claim to have made any striking advances in the subject, although we make comments on one or two conclusions by earlier writers.

2. Outline of generator

The generator is of 'shuffled' type, obtained by combining two ordinary congruential generators

$$u_{i+1} \equiv ku_i \qquad \text{(modulo } p\text{)}$$

and

$$v_{i+1} \equiv lv_i + 1 \quad \text{(modulo } q\text{)}$$

where k, l, p, q, u_i and v_i are integers.

The u sequence provides the numbers actually given to the user; the v sequence decides the order in which they are presented. The purpose of the shuffle is to reduce the risk of correlations between nearby members of the generated sequence, a problem which frequently occurs with single congruential generators. The choice of p and q was determined largely by machine considerations, to avoid division instructions. If $p = 2^n$ the formation of remainders is trivial, but the resulting numbers have a short-period cyclic behaviour at the less significant end; if $p = 2^n \pm 1$ the formation of remainders is almost as easy and there is no regular bottom-end behaviour. In particular $2^{31} - 1$ is a prime, so that there exist values of k such that u_i covers all integers between 1 and $p - 1$ before repeating itself. We therefore took this value for p; the choice of k is discussed in Section 4. The choice of q is easier, since short-period cyclic behaviour of the bottom end of the v sequence does not matter; we therefore took $q = 2^{32}$ as being the simplest value for the machine, and took $l = 2^7 + 1$, which has been quoted as giving a 'satisfactory' sequence for $q = 2^{35}$, so that we might expect it to be satisfactory for our limited purposes with $q = 2^{32}$. We list all our sources in Section 5 at the end, for convenience.

The tactical details are straightforward. The generator contains an internal table with space for 128 integers. On an 'initialisation' entry, which must always be made before the user attempts to form any numbers, the user supplies some value u_0 satisfying $1 \leqslant u_0 \leqslant 2^{31} - 2$ and the routine fills the table with u_1 to u_{128}; it also sets v_0 to a standard value. Subsequent entries to the generator may obtain numbers in floating-point or integral form, a prime intention being to produce them in forms suitable for use in high-level languages. For a floating number, the routine obtains the next member of the v sequence and uses its top seven bits to indicate an element of the table; it scales this number by 2^{-31}, giving a result in the range 0 to 1, and puts the next member of the u sequence into the table instead of it. For an integer, the user supplies a value n; the routine forms an intermediate result in the range 1 to $2^{31} - 2$ as above and then multiplies it by $n.2^{-31}$ discarding the fractional part, giving a result with an almost exactly uniform distribution in the range 0 to $n - 1$ inclusive. Notice that n may be changed on each entry. (Because the u sequence cannot produce the values 0 or $2^{31} - 1$ the floating numbers produced cannot take the values $0, 1 - 2^{-31}$ or 1; apart from this the distribution is uniform in the range 0 to 1, and the edge effect is of no practical importance. Similar unimportant departures occur with the integral distributions. There are actually two forms of floating point number on System 4: 'long' (56-bit mantissa) and 'short' (24-bit mantissa). For 'short' numbers one can simply discard the bottom seven bits, and for 'long' ones it is worth remembering that the bottom 25 bits will be zeros; this is rarely of importance —if it is the number may be padded out with 25 bits from a second random number—and the extra generation time for a 56-bit generator did not appear to be justified.)

3. Testing strategy

The intention of the testing strategy was to make reasonably certain that the routine was fit for publication without spending a disproportionate amount of effort on this particular software item. We had an existing KDF9 program which had proved effective in the past at sorting out inferior generators, and we therefore decided to code the System 4 generator for KDF9 and use this existing test program either unchanged or with only minor modifications; in the event the minor modifications proved to have no particular worth and were dropped for some of the later runs. The only testing done on a System 4 computer was to ensure that the sequence produced on it was correct and the same as that produced on KDF9. There is always the risk that the next test tried on a hitherto satisfactory generator will blow it sky-high; there also comes a time when effort that might be spent on providing another fresh test will probably be more profitable if released for use on some other software problem.

* *International Computers Limited, Kidsgrove, Staffordshire*

The main tests used were similar to those used by MacLaren and Marsaglia (1965), and in their standard form were as follows:

1. Uniformity on a line (128 cells): each number was obtained as an integer in the range 0 to 127, and a count was made of the numbers falling in each cell; the χ^2 statistic was then computed and tabulated, together with the probability of exceeding the tabulated value at random. This is of course the interesting figure, and should itself show a random variation over a large sample.
2. Uniformity on a square (16 cells each way): each number is obtained as an integer in the range 0 to 15, and each pair of numbers defines one point (so that $2n$ are needed to define n points); the χ^2 statistic is computed as before. This and the next five tests are designed to show up correlations between adjacent or nearby numbers.
3. Uniformity on a cube (8 cells each way): similarly, with $3n$ numbers defining n points.
4. Maximum of two on a line (32 cells): the larger member of each pair was taken, $2n$ numbers defining n points.
5. Minimum of two on a line (32 cells): similarly.
6. Maximum of three on a line (32 cells): similarly, $3n$ numbers defining n points.
7. Minimum of three on a line (32 cells): similarly.

MacLaren and Marsaglia also used some 'sums' tests, which we did not copy because they were satisfied, in their paper, by some generators which did not satisfy the tests above.

In all the above tests 8,192 points were taken; in the tests for the maximum and minimum of three the three least popular cells were counted as one for the purposes of the χ^2 test. The tests were performed in succession, different numbers being used for each test. The probabilities were calculated by assuming $\sqrt{(2\chi^2)}$ to have a normal distribution with unit variance and mean $\sqrt{(2\nu - 1)}$ where ν (large) is the number of degrees of freedom; comparison between this approximation and the table given in 'Cambridge Elementary Statistical Tables' showed it to be quite adequate for our requirements. From time to time we tried these tests with different cell sizes and other variations; these never showed any significantly different behaviour from the set above, and for ease of comparison between different generators we ultimately settled on this set of tests as a standard.

The probabilities thus calculated were expressed as percentages, and are denoted by '$P(\chi^2)$' in the tables following. A run in order through the above set was called a 'cycle', and the basic pattern was to initialise the generator using some value u_0 and run one or more cycles, reinitialise with a new u_0 and repeat, and so on.

4. Generators tested

The first generator tried used $k = 2^{27} - 1$, with the shuffle, and was tried with the following tests:

1. Sequence as above, $u_0 = 1(1)75$, one cycle each;
2. Sequence as above, $u_0 = 24,25(25)100$, five cycles each;
3. Various other tests involving distribution within rectangles, cuboids and four-dimensional volumes of various sizes.

The results appeared essentially satisfactory; one or two bad results (e.g. four results below the 10% level out of a batch of seven, or eight below 20% out of a batch of twenty) were obtained, but there was no apparent systematic excess of bad results. (A set of 75 batches each of seven results may be expected to produce at least one batch with four or more results below the 10% level about one-fifth of the time. In principle, on meeting such a case one would perform similar runs ten or more times to see if the pattern is systematic; in practice the machine time required soon becomes prohibitive.)

The results were tabulated to one decimal place if below 10%, and to the nearest integer if above 10%; this aids both legibility and the detection of small values, and the suppressed digits are not of any real importance, particularly in view of the approximation made to χ^2. The original intention was to tabulate them in this form, but it soon became obvious that this would be an unacceptable use of much space, and we have therefore

Table 1

$k = 2^{27} - 1$, $p = 2^{31} - 1$ with shuffle; $u_0 = 1(1)50$, one cycle each

TEST	NUMBER OF OCCURRENCES OF $P(\chi^2)$ IN THE GIVEN RANGES (SEE SECTION 3)									
	0 to 9·95	9·95 to 19·5	19·5 to 29·5	29·5 to 39·5	39·5 to 49·5	49·5 to 59·5	59·5 to 69·5	69·5 to 79·5	79·5 to 89·5	89·5 to 100
1	3	4	2	5	6	8	6	4	6	6
2	5	3	9	5	5	3	4	7	4	5
3	4	6	2	5	4	5	5	6	6	7
4	3	4	7	7	7	6	2	3	6	5
5	6	4	3	9	7	3	5	5	3	5
6	7	5	5	4	7	5	6	3	2	6
7	6	1	5	4	6	3	6	8	4	7

resorted to summarising. A copy of the full results may be had on application to either author. The obvious summaries, with blocks 0–10, 10–20 and so on, gives a slight problem with results such as '20', which give no indication of which of two adjacent cells they should be counted in, so we summarised into ten blocks with boundaries at 9·95, 19·5, 29·5, 39·5, . . ., 89·5. The blocks are thus not quite of equal size, but even if they were the probabilities would not be exactly equal due to the approximation used for χ^2—in fact the two errors cancel out to some extent; the reader can, we hope, allow for the slight distortion without difficulty. The results of the first test for $u_0 = 1(1)50$ are summarised thus in **Table 1**; this particular set appears quite typical and lines up closely with later results.

Although these results were in themselves satisfactory, Downham and Roberts (1967) had shown that some 'unshuffled' generators (i.e. generators using a u sequence only, with no internal table or shuffling v sequence) appeared satisfactory under a sequence of tests which, on our previous experience, we would have thought them likely to fail. Their generators were produced for coding in KDF9 ALGOL—they would probably not be codeable in System 4 ALGOL as they appear to involve intermediate integers outside the System 4 integer range. We therefore felt we should try their three best generators, given by

$$k = 8192, \quad p = 67101323$$
$$k = 8192, \quad p = 67099547$$
$$k = 32768, \quad p = 16775723$$

under our tests, and found them apparently satisfactory. The results for the standard test above, $u_0 = 1(1)50$ (one cycle each), for the first of these generators are summarised in **Table 2**. They are essentially typical, with the possible exception of the first line. (We have found that several generators give too few 'bad' results on the 'uniformity on a line' test, without having enough evidence to claim this as a definitely significant pattern. This particular test is of course virtually unaffected by shuffling the generator.)

As a consequence of the satisfactory behaviour of these three generators we decided to try $k = 2^{27} - 1$, $p = 2^{31} - 1$ without the shuffle, and again the results for the standard test, $u_0 = 1(1)50$, one cycle each, are given in **Table 3**. The fourth and fifth lines of this speak

Table 2

$k = 8192$, $p = 67101323$; $u_0 = 1(1)50$, one cycle each

TEST	NUMBER OF OCCURRENCES OF $P(\chi^2)$ IN THE GIVEN RANGES									
	0 to 9·95	9·95 to 19·5	19·5 to 29·5	29·5 to 39·5	39·5 to 49·5	49·5 to 59·5	59·5 to 69·5	69·5 to 79·5	79·5 to 89·5	89·5 to 100
1	0	2	4	9	5	6	7	5	7	5
2	6	5	3	7	5	4	6	7	4	3
3	6	6	4	5	5	3	3	6	4	8
4	9	3	3	6	4	2	6	3	9	5
5	5	2	3	5	7	4	7	3	8	6
6	5	3	6	7	4	6	6	4	5	4
7	4	2	9	5	7	6	4	4	7	2

Table 3

$k = 2^{27} - 1$, $p = 2^{31} - 1$; $u_0 = 1(1)50$, one cycle each

TEST	NUMBER OF OCCURRENCES OF $P(\chi^2)$ IN THE GIVEN RANGES									
	0 to 9·95	9·95 to 19·5	19·5 to 29·5	29·5 to 39·5	39·5 to 49·5	49·5 to 59·5	59·5 to 69·5	69·5 to 79·5	79·5 to 89·5	89·5 to 100
1	3	4	2	5	5	8	6	5	6	6
2	9	3	4	3	8	8	4	5	3	3
3	5	1	9	6	3	4	6	8	6	2
4	32	7	2	5	1	2	1	0	0	0
5	39	2	3	3	0	2	0	1	0	0
6	9	7	3	6	4	4	8	3	5	1
7	8	5	2	6	3	8	6	3	4	5

for themselves, and indicate that this generator is definitely unsatisfactory without the shuffle.

Some stocktaking was now needed. We had certainly failed to show the superiority of our shuffled generator over the Downham and Roberts unshuffled ones. However it is a property of unshuffled generators that there are always values of k for which their behaviour is unsatisfactory, and if, for some n not impractically large, k^n is such a value then the generator using multiplier k may produce successive n-tuples of numbers which are not independent of each other. This cannot be tested satisfactorily without tests on n-tuples, and the time taken for such testing for all reasonable n is prohibitive. If, therefore, the user has a problem requiring the use and relative independence of n-tuples, and the tests quoted for the generator do not include tests for this value of n, he ought for safety to do his own. The 'shuffle' substantially reduces, perhaps to vanishing point, the risk resulting from not doing this, so that it remains firmly our opinion that a generator intended for standard library use should include a shuffle. There is, however, no good reason for using a proven bad value of k, such as $2^{27} - 1$ above, if a better one can be found. We therefore tried another value from the literature, $k = 13^{13} = 455470314$ modulo $2^{31} - 1$, and ran our standard test on the unshuffled generator for $u_0 = 1(1)50$, one cycle each, obtaining satisfactory results, as we did from the same test with $k = 13^{26}$, $k = 13^{39}$ and $k = 13^{52}$ (which correspond respectively to taking $u_2, u_4, u_6 \ldots$, $u_3, u_6, u_9 \ldots$ and $u_4, u_8, u_{12} \ldots$ from the generator given by $k = 13^{13}$, thus getting at least some indication of possible correlations between successive n-tuples for $n > 3$, a simple technique which is of course not possible with shuffled generators).

On this evidence we regarded $k = 13^{13}$ as a clearly better choice than our original $k = 2^{27} - 1$, and modified the generator accordingly. We then ran one cycle of the standard test for $u_0 = 1(1)100$, and ten cycles for $u_0 = 1(1)10$, including the shuffle in each case, giving results which are summarised in **Tables 4 and 5**. The first cycle of each group of ten has been ignored in Table 5, since it already contributes to Table 4. The values 2 and 19 in the first column of Table 4 were both unexpected, but in all the circumstances we are inclined to regard them as casual curiosities rather than genuine trends; the machine time needed to repeat the runs

Table 4

$k = 13^{13}$, $p = 2^{31} - 1$ with shuffle; $u_0 = 1(1)100$, one cycle each

TEST	NUMBER OF OCCURRENCES OF $P(\chi^2)$ IN THE GIVEN RANGES									
	0 to 9·95	9·95 to 19·5	19·5 to 29·5	29·5 to 39·5	39·5 to 49·5	49·5 to 59·5	59·5 to 69·5	69·5 to 79·5	79·5 to 89·5	89·5 to 100
1	7	10	11	13	9	10	8	9	10	13
2	10	12	12	15	17	3	6	11	10	4
3	6	10	16	11	9	11	15	7	9	6
4	2	8	5	12	12	14	13	14	11	9
5	19	10	10	9	10	12	9	12	5	4
6	15	8	10	10	10	10	12	4	11	10
7	14	9	6	6	9	11	9	12	13	11

Table 5

$k = 13^{13}$, $p = 2^{31} - 1$ with shuffle; $u_0 = 1(1)10$, ten cycles each omitting the first cycle of each group of ten

TEST	NUMBER OF OCCURRENCES OF $P(\chi^2)$ IN THE GIVEN RANGES									
	0 to 9·95	9·95 to 19·5	19·5 to 29·5	29·5 to 39·5	39·5 to 49·5	49·5 to 59·5	59·5 to 69·5	69·5 to 79·5	79·5 to 89·5	89·5 to 100
1	10	13	9	11	8	10	9	6	9	5
2	11	5	10	6	9	12	10	7	13	7
3	12	2	6	11	11	9	9	8	13	9
4	7	9	8	10	10	8	8	10	10	10
5	9	6	10	10	11	10	12	9	5	8
6	5	5	13	10	11	10	7	7	5	17
7	15	9	6	9	9	4	11	12	6	9

enough times to prove this is quite unavailable at present. We therefore regarded the shuffled generator with $k = 13^{13}$, $p = 2^{31} - 1$ as, on the strength of these results, fit for general release.

5. Sources

A good deal of interesting information is contained in a descriptive paper by Hayes (1960), which also contains a substantial list of early references. Hayes quotes Greenberger (1959) for $k = 2^{27} - 1$, $p = 2^{31} - 1$ and Edmonds (1960) for $k = 13^{13}$, $p = 2^{31} - 1$; and Rotenberg (1960) for $l = 2^7 + 1$, $q = 2^{35}$. The value 17596 68861 for v_0 is a sequence of random numbers from Cambridge Elementary Statistical Tables.

6. Conclusions

The immediate and most relevant conclusion is that the S4 generator is, in our opinion, fit for general release. This does *not*, in passing, mean that it will produce systematically 'nice' results; in fact we hope it will produce about the correct number of 'bad' results. A random number generator produces unpredictable numbers, not 'nice' numbers, a point the user must always bear in mind. If, however, any test is found on which this generator gives significantly and systematically bad results we would like to know.

Some slight comments on our sources may be worth making. Hayes quotes Greenberger as stating, though without giving details, that the generator $k = 2^{27} - 1$, $p = 2^{31} - 1$ 'has passed the standard frequency and correlation tests', which cannot be maintained in the light of our results. Obviously Greenberger's standard tests did not include our 'maximum' and 'minimum' tests, but we remarked earlier that the risk always exists that the next test tried will blow a generator sky-high. Hayes had some criticism of $l = 2^7 + 1$, $q = 2^{35}$ ('Tests have been carried out, with satisfactory results, for $a = 7$. . . . A larger value of a would seem preferable, however, from a statistical point of view', 'a' here being the power of 2 used in forming l); we did not ignore this, but felt that as it was only used for the shuffle mild bad behaviour would not seriously affect the final generator.

Our results here do not themselves provide a strong case for abandoning single congruential generators, in the way that MacLaren and Marsaglia's—or for that matter our own earlier experience—seemed to. However it remains our opinion that a single congruential generator throws a greater burden on the user to verify the adequacy of the numbers for his particular application, so that we definitely think the shuffle should not be omitted from a standard library routine even though it costs some space and a little time; reliability is more important than either of these.

Our final remarks concern the testing procedures which should be used on similar generators to these. The approach used here has one serious disadvantage—there is no obvious criterion which tells one where to stop. For single congruential generators Coveyou and Macpherson (1967) propose quite a different approach based on Fourier analysis. For more complicated generators, as here, no similar approach is known to us, and one is therefore forced back entirely upon tests similar in spirit to the above. The extension of *a priori* methods, including Fourier methods, to multiple congruential generators could be a substantial advance; it is also not obvious how this could be done in practice.

Acknowledgements

Our thanks are due to the referee for several valuable suggestions on detail, and to International Computers Ltd. for permission to publish this paper.

References

COVEYOU, R. R., and MACPHERSON, R. D. (1967). Fourier analysis of uniform random number generators, *JACM*, Vol. 14, p. 100.

DOWNHAM, D. Y., and ROBERTS, F. D. K. (1967). Multiplicative congruential pseudo-random number generators, *The Computer Journal*, Vol. 10, p. 74.

EDMONDS, A. R. (1960). The generation of pseudo-random numbers on electronic digital computers, *The Computer Journal*, Vol. 2, p. 181.

GREENBERGER, M. (1959). Random number generators, Preprint for 14th National Meeting of the ACM, p. 1.

HAYES, J. G. (1960). Methods of generating random numbers, DEUCE News 52, p. 3.

LINDLEY, D. V., and MILLER, J. C. P. (1952). *Cambridge Elementary Statistical Tables*, Cambridge University Press.

MACLAREN, M. D., and MARSAGLIA, G. (1965). Uniform random number generators, *JACM*, Vol. 12, p. 83.

ROTENBERG, A. (1960). A new pseudo-random number generator, *JACM*, Vol. 7, p. 75.