Programming Techniques

R. M. McCLURE, Editor

Coding the Lehmer Pseudo-random Number Generator

W. H. PAYNE, J. R. RABUNG, AND T. P. BOGYO
Washington State University, Pullman, Washington

An algorithm and coding technique is presented for quick evaluation of the Lehmer pseudo-random number generator modulo 2 ** 31 − 1, a prime Mersenne number which produces 2 ** 31 − 2 numbers, on a p-bit (greater than 31) computer. The computation method is extendible to limited problems in modular arithmetic. Prime factorization for 2 ** 61 − 2 and a primitive root for 2 ** 61 − 1, the next largest prime Mersenne number, are given for possible construction of a pseudo-random number generator of increased cycle length.

KEY WORDS AND PHRASES: pseudo-random number, random number, modular arithmetic, uniform probability density, uniform frequency function, simulation, prime factorization, primitive roots
CR CATEGORIES:

Most pseudo-random number generators are of the type suggested by Lehmer,

$$X_{n+1} \equiv KX_n (\text{mod } m) \tag{1}$$

where the modulus m is chosen as 2^{p-1} for a p-bit-word binary machine. Selection of this particular modulus avoids the division necessary for general modular arithmetic, thus speeding actual computation. Such a composite modulus is unsatisfying from a number theory point of view, however. The cycle length, which depends on the starting point, cannot be greater than 2^{p-3}. Most of these generators produce only odd numbers [1]. For example, the multiplier $K = 2^{18} + 3$, $m = 2^{p-1}$ and X_0 odd, produces only half the odd digits. The particular half produced is determined by X_0.

If m is of the type q^n or $2q^n$, q an odd prime, then K can be selected to be a primitive root for the modulus. Lehmer [2, 3] suggested $K = 14^{29}$, $m = 2^{31} - 1$ (a prime Mersenne number), and $0 < X_0 < 2^{31} - 1$. Fourteen was chosen because it is a primitive root modulo $2^{31} - 1$; a multiplicative group of order

$$2^{31} - 2 = 2 \times 3^2 \times 7 \times 11 \times 31 \times 151 \times 331 = 2{,}147{,}483{,}646 \text{ (cycle length)}$$

was generated. Raising 14 to the vth power where v is relatively prime to the cycle length gives another primitive root modulo $2^{31} - 1$. Choosing $v = 29$ introduces "randomness" into the number sequence.

We present an algorithm and coding technique for more rapid evaluation of the Lehmer pseudo-random number generator with limited extensions to modular arithmetic problems. Begin with

$$X_{n+1} \equiv KX_n (\text{mod } 2^{p-1}) \tag{2}$$

or, by definition,

$$X_{n+1} = KX_n - r2^{p-1} \tag{3}$$

for some r. Adding r to both sides of eq. (3) yields

$$X'_{n+1} = X_{n+1} + r \equiv KX_n (\text{mod } 2^{p-1} - 1) \tag{4}$$

provided that $X_{n+1} + r < 2^{p-1} - 1$ (no overflow). If $X_{n+1} + r > 2^{p-1} - 1$ (overflow), then

$$X'_{n+1} \equiv X_{n+1} + r + 1 (\text{mod } 2^{p-1}), \tag{5}$$

which leads directly to

$$X'_{n+1} \equiv KX_n (\text{mod } 2^{p-1} - 1). \tag{6}$$

This method is extendible to any modulus $m = 2^{p-1} + d$,

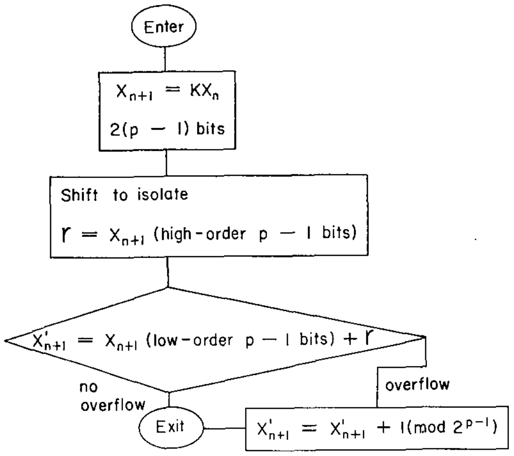

Enter
$X_{n+1} = KX_n$
2(p − 1) bits
Shift to isolate
$r = X_{n+1}$ (high-order p − 1 bits)
$X'_{n+1} = X_{n+1}$ (low-order p − 1 bits) + r
no overflow
overflow
Exit
$X'_{n+1} = X'_{n+1} + 1 (\text{mod } 2^{p-1})$

FIG. 1

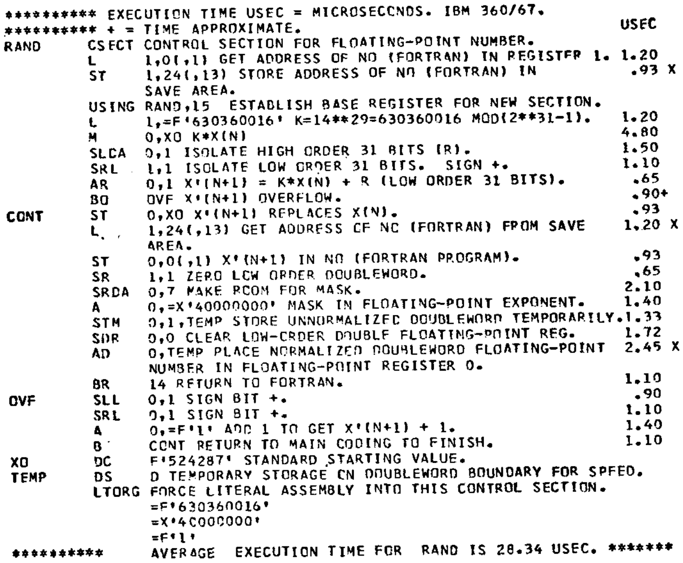
but is efficient only if d is a power of two. A modified version of this method was suggested by Lehmer for use on ENIAC. However, it did not use the overflow feature

$$U_{n+1} \equiv 23U_n(\text{mod } 10^8 + 1 = 17 \times 5{,}882{,}353), \qquad U_j = 8 \text{ decimal digits.}$$

The flowchart in Figure 1 illustrates the algorithm.

Evaluation of eq. (1) with $m = 2^{31}$ takes 8.65 μsec; for $m = 2^{31} - 1$, 12.40 μsec (average) using this algorithm, and 14.70 μsec performing the division (IBM 360/67 instruction times). Additional time savings for the algorithm can be realized on a computer developing $2(p - 1)$ product digits (even though $p > 31$) rather than the $2p - 1$ product digits developed on an IBM 360.

In coding this algorithm care must be taken when converting the fixed-point results to floating-point No low order zeros should be introduced, thereby biasing small values when normalizing the result. A section of possible coding for the floating-point pseudo-random number on an IBM 360 is given in Figure 2.



```
*********** EXECUTICN TIME USEC = MICROSECCNDS. IBM 360/67.
*********** + = TIME APPROXIMATE.                                  USEC
RAND     CSECT CONTROL SECTION FOR FLOATING-POINT NUMBER.
         L     1,0(,1) GET ADDRESS OF NO (FORTRAN) IN REGISTFR 1. 1.20
         ST    1,24(,13) STORE ADDRESS OF NO (FORTRAN) IN          .93 X
               SAVE AREA.
         USING RAND,15  ESTABLISH BASE REGISTER FOR NEW SECTION.
         L     1,=F'630360016' K=14**29=630360016 MOD(2**31-1).   1.20
         M     0,XO K*X(N)                                        4.80
         SLCA  0,1 ISOLATE HIGH ORDER 31 BITS (R).                1.50
         SRL   1,1 ISOLATE LOW ORDER 31 BITS.  SIGN +.            1.10
         AR    0,1 X'(N+1) = K*X(N) + R (LOW ORDER 31 BITS).       .65
         BO    OVF X'(N+1) OVERFLOW.                               .90+
CONT     ST    0,XO X'(N+1) REPLACES X(N).                         .93
         L     1,24(,13) GET ADDRESS CF NC (FORTRAN) FPOM SAVE    1.20 X
               AREA.
         ST    0,0(,1) X'(N+1) IN NO (FORTRAN PROGRAM).            .93
         SR    1,1 ZERO LCW ORDER DOUBLEWORD.                      .65
         SRDA  0,7 MAKE RCOM FOR MASK.                            2.10
         A     0,=X'40000000' MASK IN FLOATING-POINT EXPONENT.    1.40
         STM   0,1,TEMP STORE UNNORMALIZEC DOUBLEWORD TEMPORARILY.1.33
         SDR   0,0 CLEAR LOW-CRDER DOUBLF FLOATING-POINT REG.     1.72
         AD    0,TEMP PLACE NORMALIZED DOUBLEWORD FLOATING-POINT  2.45 X
               NUMBER IN FLOATING-POINT REGISTER 0.
         BR    14 RFTURN TO FORTRAN.                              1.10
OVF      SLL   0,1 SIGN BIT +.                                     .90
         SRL   0,1 SIGN BIT +.                                    1.10
         A     0,=F'1' ADC 1 TO GET X'(N+1) + 1.                  1.40
         B     CCNT RETURN TO MAIN CODING TO FINISH.              1.10
XO       DC    F'524287' STANDARD STARTING VALUE.
TEMP     DS    D TEMPORARY STORAGE CN DOUBLEWORD BOUNDARY FOR SPFED.
         LTORG FORCE LITERAL ASSEMBLY INTO THIS CONTROL SECTION.
               =F'630360016'
               =X'4C000000'
               =F'1'
***********    AVERAGE  EXECUTION TIME FOR  RAND IS 28.34 USEC. *******
```

FIG. 2

Since it is possible to generate approximately 2,000,000 fixed-point pseudo-random numbers per minute on a moderately fast (e.g. IBM 360/67) computer, a known long cycle length is important. All numbers excluding 0 modulo $2^{31} - 1$ appear in this sequence. Thus no obvious degradation in the low order digits occurs, as when using a generator with a composite modulus. Thus in choosing a random integer, I, $1 \leq I \leq N$, a standard in-line modular arithmetic subroutine can be entered with a fixed-point pseudo-random number argument and one added to the output. Faster yet the high order bits may be used by multiplying the fixed-point value by N and adding one to the high order 31 bits. Of course the problem of accidentally choosing a starting value which leads to a short sequence is entirely avoided.

The sequence of pseudo-random numbers modulo $2^{31} - 1$ passed a number of statistical tests. The uniformity of the distribution was tested by dividing the interval 0 to 1 into 1024 subintervals and calculating a χ^2 value with 1023 d.f. In the serial test the interval was subdivided into 32 subintervals and a 32×32 matrix was formed. The cells represented the number of numbers in the ith interval followed by a number in the jth interval. Again a χ^2 value was used to test the uniformity of this distribution. The frequency of length of runs was tested by the occurrence of $k + 1$ consecutive numbers forming a monotonic increasing (or decreasing) sequence which was broken by the $(k + 2)$-th number and was called a run of length k. Comparing the observed frequencies of lengths of run with expected values that can be calculated lends itself to a χ^2 statistic to be used as a test of randomness. Besides these tests, several million uniformly distributed pseudo-random numbers were generated, and it was found that the mean of these numbers was 1/2 and the variance was .0833, as was expected.

The time required to evaluate this algorithm, the time spent each year generating pseudo-random numbers at a computer installation, and the advance knowledge of the output of the Lehmer generator make this particular method attractive.

The anticipated increase in computer speed and the fact that the next prime Mersenne number is $2^{61} - 1$ combine to present an intriguing design and coding problem for a pseudo-random number of cycle length

$$2^{61} - 2 = 2 \times 3^2 \times 5^2 \times 7 \times 11 \times 13 \times 31 \times 41 \times 61 \times 151 \times 331 \times 1321 = 2{,}305{,}843{,}009{,}213{,}693{,}950.$$

This generator must, of course, be tested for "randomness" using the primitive root of 37, some powers of which provide candidates for K (mod $2^{61} - 1$).

Acknowledgment. Appreciation is expressed to D. H. Lehmer for providing a primitive root for $2^{61} - 1$.

RECEIVED AUGUST, 1968

REFERENCES

1. VAN GELDER, A. Some new results in pseudo-random number generation. *J. ACM 14*, 4(Oct. 1967), 785–792.
2. LEHMER, D. H. Random number generation on the BRL high-speed computing machines, by M. L. Juncosa. *Math. Rev. 15* (1954), 559.
3. TOCHER, K. *The Art of Simulation.* English U. Press, London, 1963.
4. Random number generation and testing. Form C20-8011, IBM, 1959.