NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR PACKAGE LLRANDOM

Gerard P. Learmonth, et al

Naval Postgraduate School
Monterey, California

June 1973
NAVAL POSTGRADUATE SCHOOL
Monterey, California

RANDOM NUMBER GENERATOR PACKAGE LLRANDOM
by
G. P. Learmonth
and
P. A. W. Lewis
June 1973

Approved for public release; distribution unlimited.
This report is intended to describe an interim version of a computer program package for random number generation on the IBM System/360. The package, when called by a FORTRAN IV program, will deliver either a single value or an array (of specified size) of single precision uniformly, normally, or exponentially distributed pseudo-random deviates, or a single value or an array of uniformly distributed integers between 1 and $2^{31}-1$. The package also has the ability (optional) to "shuffle" the pseudo-random numbers to obtain "better" statistical properties.
Random number generator
Pseudo-random numbers
Normal distribution
Exponential distribution
Shuffled random numbers
Division simulation
Congruential generator
NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral M. B. Freeman
Superintendent

M. U. Clauser
Provost

ABSTRACT

This report is intended to describe an interim version of a computer program package for random number generation on the IBM System/360. The package, when called by a FORTRAN IV program, will deliver either a single value or an array (of specified size) of single precision uniformly, normally, or exponentially distributed pseudo-random deviates, or a single value or an array of uniformly distributed integers between 1 and $2^{31}-1$. The package also has the ability (optional) to "shuffle" the pseudo-random numbers to obtain "better" statistical properties.

Prepared by:

Peter A. W. Lewis*
Department of Operations Research and Administrative Sciences

Approved by:

Gerard P. Learmonth
Computer Center

Released by:

J. H. Wozencraft
Dean of Research

*Partial support by the Office of Naval Research through Contract NRO42-284 is gratefully acknowledged.
NAVAL POSTGRADUATE SCHOOL
RANDOM NUMBER GENERATOR PACKAGE LLRANDOM

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Some definitions</td>
<td>1</td>
</tr>
<tr>
<td>II.</td>
<td>The Generator</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A. Division simulation</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>B. Shuffling</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>C. Uniform (0.0,1.0) random numbers</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>D. Normally distributed random deviates</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>E. Exponential distribution</td>
<td>12</td>
</tr>
<tr>
<td>III.</td>
<td>HOW TO USE THE PACKAGE</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>A. Implementation</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>B. Future Enhancements</td>
<td>20</td>
</tr>
</tbody>
</table>

References 21

Appendix of Program Listings
I. Introduction.

Numerous random number generators have been proposed for the System/360. Several of these generators have been incorporated into the subroutine library here at the Computer Center. The adequacy of some of these generators has rested on the results of some rather weak tests for randomness; recent results in the literature have shown many of these generators to be very poor performers. This report will describe an interim version of a package for random number generation which has stood up under intensive statistical testing and is deemed to be very satisfactory for the System/360. (The statistical testing will be reported elsewhere.)

The package, when called by a FORTRAN IV program, will deliver either a single value or an array (of specified size) of single precision, uniformly, normally, or exponentially distributed pseudo-random deviates or a single value or an array of pseudo-random integers uniformly distributed between 1 and $2^{31}-1$. The package also has the ability (optional) to "shuffle" the pseudo-random numbers to obtain "better" statistical properties.

Further refinements will be made to this generator; however, it is now available for use in an interim version under the name LLRANDOM. Future versions will be announced through the W. R. Church Computer Center Newsletter. The changes envisioned will be internal and aimed at increasing speed and efficiency of coding. The actual numbers produced in future versions will remain the same as described here, as will the FORTRAN calling sequence.

Some definitions. By "random number generator" or "pseudo-random number generator" is meant an algorithm by which sequences of numbers
are produced which follow a given probability distribution and possess the appearance of randomness. Without attempting to address the still unresolved philosophical question of what a random sequence is, the underlined words above are the keys to random number generation on a digital computer. The term sequence implies that many random numbers must be produced serially from the algorithm. The user may need only a very few of these numbers, however we generally require that the algorithm be able to produce very many numbers. Distribution implies that we can associate a probability statement with the occurrence of each random number. The distribution is usually taken to be uniform, that is, within a given range the probability of occurrence of a given number is the same as for any other number in a similar range. If the algorithm produces, say, m distinct numbers then the probability of occurrence for any one of them is 1/m.

Lastly, we speak of the appearance of randomness. As will be shown next, the actual implementation of the algorithm is a recurrence relation where each succeeding number is a function of the preceding number. True randomness would require independence of successive numbers; however, the algorithm generates a deterministic sequence. Algorithms for random number generation do, however, yield sequences which appear to be random, hence the term "pseudo-random numbers." It is this characteristic which is the subject of statistical testing, that is, one asks, "how random does the given sequence appear?"

The uniform random number generator which forms the basis of the package described here is a Lehmer congruential generator. The recurrence relation is given by
This generator produces integer random numbers between 1 and m. These integer values may then be transformed into real-valued numbers between 0.0 and 1.0 or into any desired distribution by an appropriate transformation.

II. The Generator.

The recurrence relation given in equation (1) is actually called a "Lehmer mixed congruential generator." The term mixed comes from the fact that it involves a multiplication by a constant, A, plus an addition of a constant, C. The actual implementation used in LLRANDOM is called a "multiplicative," or "pure," congruential generator in that we take C = 0, giving

$$X_n = A \cdot X_{n-1} \pmod{m}.$$  \hfill (2)

The field of positive integers is, of course, infinite. It is a reality of digital computers that only a finite number of positive integers are expressible. Specifically, we are limited to the word size of the System/360. This word size is 32 bits with one bit reserved for the algebraic sign; hence, an obvious choice for m is $2^{31}$. The product $A \cdot X_{n-1}$ is formed by the System/360 in two adjacent registers yielding a result which may be as large as $2^{63}$. We must, however, reduce this product to a number less than or equal to $2^{31}$. The mod, or modulo, operation accomplishes this. The product $A \cdot X_{n-1}$ is divided by $2^{31}$ leaving a quotient which is some multiple of $2^{31}$ and a remainder which is strictly less than $2^{31}$. It is this remainder which is the next pseudo-random number $X_n$ in the equation (1).
On first examination it would appear that a full $2^{31}$ numbers could be generated by the sequence (1). This is not the case, unless $A$ and $m$ are chosen properly. We define a term called the period which is the number of unique random numbers computable for a given choice of $A$ and $m$. To illustrate the concept, assume we have a six-bit word with one bit for a sign. We then have $m = 2^5 = 32$.

Choose $A = 9$ and work through a sequence starting with $X_0 = 1$.

<table>
<thead>
<tr>
<th>Step $n$</th>
<th>$X_{n-1}$</th>
<th>$A \cdot X_{n-1}$</th>
<th>$A \cdot X_{n-1} \pmod{2^5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>81</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>153</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>225</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Note that the modulus of this generator is 32, however we have realized a period of only 4, that is the sequence of 1, 9, 17, 25, 1, 9, ... repeats after only 4 numbers. Obviously, care must be taken to insure that such occurrences do not happen in a random number generator. Hopefully, the period will also be independent of the starting value, $X_0$.

A great deal of work has been done on number theoretic considerations for the choice of $m$ so as to yield a maximum period length (see Knuth\(^3\)). To summarize, generators with modulus $m = 2^p$ for any integer, $p$, can have a maximum period of $m/4$, or, for the System/360, $2^{31}/4 = 2^{29}$; the period may also depend on the starting value. When the modulus $m$ is a prime number, the maximum possible period is $m - 1$. 
It so happens that the largest prime less than or equal to $2^{31}$ is $2^{31} - 1$, which is most fortuitous. Hence, choosing $m = 2^{31} - 1$ we can achieve a maximum period of $m - 1 = 2^{31} - 2$. These results produce only upper bounds on the period length. Recall in the example above, the maximum period possible is $2^5/4 = 2^3 = 8$, but that a period of only 4 was observed. This naturally leads to considerations of the choice of the multiplier, A.

Success in achieving a maximum period lies with the choice of the multiplier. Again, to briefly summarize the pertinent number theory, for a modulus $2^{31}$ the multiplier A must differ by 3 from the nearest multiple of 8; the starting value, $X_0$, must be odd; A must be one greater than a multiple of 4; and C must be odd. These conditions only assure a maximum period of $m/4$, not necessarily good statistical properties. For the random number generator described here (LLRANDOM) we are choosing $C = 0$; hence, this length is not achievable if $m = 2^{31}$. Luckily, the conditions on choosing A for the modulus $m = 2^{31} - 1$ are more easily met and we can achieve the maximum period.

Utilizing some of the nice number theoretic properties of the number $2^{31} - 1$, to achieve a maximum period, A must be a positive primitive root of $2^{31} - 1$ or a power of such a number. This is generally not easy to find; the value of A used in the generator described here is $7^5$. The number 7 is a positive primitive root of $2^{31} - 1$ and raising 7 to the fifth power results in the multiplier 16807 which is also a positive primitive root of $2^{31} - 1$ (Lewis, Goodman, and Miller) and satisfies some conditions regarding the statistical performance of the generated sequence. These conditions will not be discussed here.
The generator

\[ X_n \equiv 7^5 \cdot X_{n-1} \pmod{2^{31}-1} \]  

(3)

is the generator reported in Lewis, Goodman, and Miller\(^1\). The authors cite the results of very extensive tests on this generator, all of which show that it is very satisfactory.

A. Division simulation. A practical consideration for random number generators is that they be fast, hopefully without requiring excessive memory to achieve speed. In many applications rather large quantities of numbers are needed and the speed of the generator can be crucial.

Nearly all random number generators are coded as subroutine or function subprograms in the assembler or machine language of the computer. The algorithm for implementing (3) is rather simple, involving a multiplication and then a division to effect the modulo operation. On most computers the division operation is rather slow as compared to the multiplication operation. In the past, the multiplier \( A \) was chosen so that its binary representation contained many zeroes, thereby speeding the multiplication. Unfortunately, this choice was at the expense of the period length, since such multipliers rarely met the number theoretic conditions for a maximum period. For the LLRANDOM generator (3) described here, the division operation has been replaced by a division simulation involving two shifts and an add instruction. Should a fixed-point (integer) overflow occur, two more additions are required to correct the situation.

The ordinary division on a System/360 Model 67-2 requires 8.49 micro-seconds. Without overflow, the simulation requires only 3.45
micro-seconds. When overflow occurs, the simulation takes an additional
2.32 micro-seconds for a total of 5.77 micro-seconds. These overflows
occur quite rarely, on the order of only once in 250,000 iterations.

The division simulation algorithm (again due to Lehmer) is di-
cussed by Payne, Rabung, and Bogyo and works as follows. Define a
congruence relationship by

\[ X'_n \equiv A \cdot X_{n-1} \pmod{2^{31}}. \]  \hspace{1cm} (4)

Performing the modulo operation on the product \( AX_{n-1} \) would give

\[ AX_{n-1} = q2^{31} + r, \]  \hspace{1cm} (5)

where \( q \) is some quotient and \( r \) is the remainder and is strictly less
than \( 2^{31} \). Adding \( q \) to both sides of (4) we get

\[ X_n = X'_n + q \equiv AX_{n-1} \pmod{2^{31}-1}. \]  \hspace{1cm} (6)

This form gives the desired modulus of \( 2^{31} - 1 \), if there is no overflow
in the addition of \( X'_n + q \). If there is overflow, to correct it we
merely add a constant of 1 to get

\[ X_n \equiv X'_n + q + 1 \pmod{2^{31}} = A \cdot X_{n-1} \pmod{2^{31}-1}, \]  \hspace{1cm} (7)

which is again, the desired result.

This division simulation algorithm is very easily implemented
on the System/360 and saves considerable execution time over conventional
division.

B. Shuffling. The sequence produced by the generator (3) does appear
to consist of independent, uniformly distributed numbers for most purposes.
We realize that the numbers are not actually independent, due to the procedure used to generate them. It has been proposed that a sequence of such numbers be further randomized, or "shuffled," to improve upon the appearance of randomness (see, for instance, Knuth\textsuperscript{3}). Serial correlation tests are usually employed to detect lack of independence in a sequence and at least one generator, RANDU, known to perform badly in a three-dimensional serial test was improved by shuffling. These tests will be discussed elsewhere. The various shuffling procedures which have been put forward have had little empirical validation.

The package described here has a built-in shuffling mechanism and it works as follows. A table of 128 random integers is maintained in the package. The starting values in the table represent members of the sequence (3) lagged by one million integers starting with an arbitrary seed. When a new integer is generated by the algorithm, its right-most seven bits are masked-off to form an index into the table \((2^7=128)\). The integer in the table indexed by the right-most seven bits is returned to the caller and that table entry is replaced by the integer just generated. In essence, we are taking "chunks" of 128 numbers from the basic sequence and shuffling them before they are used.

This particular shuffling scheme is dependent on the choice of the modulus. For a modulus of \(2^{31}\) the right-most bits of a congruential random number generator are non-random and their use in this scheme would defeat the purpose of shuffling. However, with a modulus of \(2^{31} - 1\) and the positive primitive root multiplier \(A = 7^5\), the right-most bits are quite random and the desired results are obtained.
C. Uniform (0.0,1.0) random numbers. So far, we have discussed how to generate uniform random integers over the range 1 to \( m = 2^{31} - 1 \). In most applications, uniform random numbers over the range 0.0 to 1.0 are desired. In theory, the uniform integers, \( X_i \), are divided by \( m \) to produce these numbers, as

\[
U_i = \frac{X_i}{m}. \tag{8}
\]

In actual implementation on the System/360, the integer result is algebraically shifted right seven bits and a normalized floating point exponent is logically OR'ed on to it. The result is a properly normalized floating point random number over the range 0.0 to 1.0, usually referred to as a "real" uniform number.

D. Normally distributed random deviates. The uniformly distributed random numbers described above are not only useful in their own right, but form the basis of transformations into random numbers with other probability distributions. One of the most important of these distributions is the Normal distribution.

There are several methods of approximating a Normal distribution with uniform random numbers. One of the oldest and, unfortunately, most common is the "sum of \( k \) uniforms method." The algorithm is based on the fact that the uniform (0.0,1.0) distribution has a mean of 1/2 and a standard deviation of \( \sqrt{1/12} \). The algorithm works as follows:

\[
X = \frac{\sum_{i=1}^{k} U_i - k/2}{\sqrt{k/12}} \tag{9}
\]
The random deviate $X$ is approximately normally distributed with mean 0 and variance 1. The approximation is not as good as other methods and it is rather time consuming in that $k$ uniforms must be generated and then summed. It was basically devised to overcome the very time consuming multiply and divide operations in older computers.

A more accurate algorithm is known as the Box-Muller method or Polar method which is actually a rejection method due to von Neumann. The method requires the generation of two uniforms to produce two independent Normals. It is based on the distribution of points inside the unit circle. The method is more accurate than the "sum of $k$ uniforms method" (in fact, theoretically perfect). However, it does require two square roots and two natural logarithm operations which are generally rather time consuming.

The algorithm used in the package described here is based on a method developed by Marsaglia and is known as the "rectangle-wedge-tail" method. This algorithm is by far the fastest algorithm available for generating normally distributed random numbers, although it requires more memory than the Polar method.

The second volume of Knuth's "The Art of Computer Programming" gives a complete and detailed description of the algorithm. Briefly, the positive half of the Normal density curve is discretized into 37 rectangles, wedges, and a tail as in Figure 1. All of the rectangles are uniformly distributed densities. The wedges are approximated by "nearly linear densities." Finally, the tail distribution is computed by a modification to the Polar method. The normal density, $f(x)$, is then given by the composite function.
f_1 - f_{12} = "large rectangles"
f_{13} - f_{24} = "small rectangles"
f_{25} - f_{36} = "wedges"
f_{37} = "tail"

FIGURE 1
RECTANGLE-WEDGE-TAIL METHOD OF APPROXIMATING THE NORMAL DENSITY
\[ f(x) = p_1f_1(x) + p_2f_2(x) + \ldots + p_{37}f_{37}(x), \] (10)

where
\[ \sum_{i=1}^{37} p_i = 1, \]

the densities \( f_1 \) to \( f_{24} \) are the rectangles; \( f_{25} \) to \( f_{36} \) are the wedges; and \( f_{37} \) is the tail. The first twelve uniformly distributed rectangles are used 88\% of the time. This makes for an extremely fast algorithm for the majority of deviates. When the tail is sampled, the deviate is generated by a modified Polar method and still quite satisfactory.

This generator for Normal deviates, like nearly all others, produces deviates with zero mean and unit variance. To change the scale and shape to any mean, \( \mu \), and the standard deviation, \( \sigma \), we apply the linear transformation
\[ Z = \mu + \sigma X \] (11)

where \( Z \) now has the desired shape and scale parameters.

E. Exponential distribution. Another probability distribution of major interest in simulations is the exponential. The cumulative distribution function and probability density function for the exponential are respectively
\[ F(x) = 1 - e^{-\lambda x}, \] (12)
\[ f(x) = \lambda e^{-\lambda x}. \] (13)

The expected value of the exponential distribution is:
\[ E[X] = 1/\lambda. \] (14)
The problem of generating exponential deviates reduces to one of generating "unit" exponentials, i.e. those with $\lambda = 1$, and then multiplying the result by whichever $\lambda$ is necessary to give the desired distribution.

One of the most common methods of generating numbers from distributions other than the uniform is to use the inverse transformation technique (see Gaver and Thompson). This can be described graphically, as in Figure 2, with a plot of the distribution function.

![Cumulative Distribution Function of the Exponential Distribution](image)

**FIGURE 2.**

**Cumulative Distribution Function of the Exponential Distribution**

The range of the abscissa, $X$, is infinite in extent. However, the range of the ordinate, $F(x)$, is $(0.0,1.0)$, the range of uniform $(0,1)$ random variables. The inverse transformation technique is to generate a uniform random number, say $U$, and use this as the ordinate. The exponential deviate, say $X$, is the abscissa point corresponding to the intersection of the ordinate and the curve.
Mathematically, this technique is expressed as

$$ u = F(x) = 1 - e^{-x}, \quad \lambda = 1, $$

$$ x = F^{-1}(u), $$

where $u$ is the uniform random number. This inverse transformation is rather easily implemented for exponentially distributed random variables via natural logarithms since we get

$$ F^{-1}(U) = -\ln(1-U), $$

or by the symmetry of the uniform distribution

$$ F^{-1}(U) = -\ln(U). $$

Perhaps the most common implementation of exponential deviate generators is this natural logarithm transformation. It is mathematically appealing as well as trivial to program, given the usual FORTRAN subroutines.

The exponential deviate generator in the LLRANDOM package is based on Marsaglia's method of dividing the probability density into a series of rectangles, wedges, and a tail. Although more complicated to program and larger in size, this method is approximately 40% faster than the logarithmic transformation.

For a survey of the generation of normal and exponentially distributed variables see Ahrens and Dieter.
III. HOW TO USE THE PACKAGE.

The random number package described here is intended solely for use on the IBM System/360 or System/370 computers. The package consists of one Assembler F control section (CSECT) with nine entry points and two FORTRAN IV function subprograms. The names of the entry points and their functions are summarized in Table 1.

The subroutine entry point OVFLOW has no calling arguments and should be called once and only once at the beginning of the user's main FORTRAN program. The function subprograms RNORTH and REXPTH are called by the Assembler routine as needed and should not be called by the user. The eight additional entry points are the names of the actual routines to generate the random numbers. There are four types of random numbers which can be generated:

(1) uniformly distributed integers on the range 1 to $2^{31} - 1$;
(2) uniformly distributed single precision floating point numbers between 0.0 and 1.0;
(3) single precision floating point normal deviates with mean zero and variance 1; and
(4) single precision floating point exponential deviates with mean 1.

There is a separate entry point for each of the four types if shuffling of the sequence is desired.

For all eight entry points, the FORTRAN calling sequence is the same, namely:

\[ \text{CALL (entry point) (IX, A, N)} \]

where

(entry point) refers to the routine desired,

viz. INT, SINT, RANDOM, SRAND, NORMAL, SNORM, EXPON, or SEXPON;
<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVFLOW</td>
<td>Calls SPIE, handles fixed point overflows. (Must be called once at start of program.)</td>
</tr>
<tr>
<td>INT</td>
<td>Generates integer random numbers.</td>
</tr>
<tr>
<td>SINT</td>
<td>Generates shuffled integer random numbers.</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Generates single precision floating point ((0.0,1.0)) random numbers.</td>
</tr>
<tr>
<td>SRAND</td>
<td>Generates single precision floating point ((0.0,1.0)) shuffled random numbers.</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Generates single precision floating point normal deviates ((\mu=0,\sigma=1)).</td>
</tr>
<tr>
<td>SNORM</td>
<td>Generates shuffled single precision floating point normal deviates ((\mu=0,\sigma=1)).</td>
</tr>
<tr>
<td>EXPON</td>
<td>Generates single precision floating point exponential deviates ((\lambda=1)).</td>
</tr>
<tr>
<td>SEXPON</td>
<td>Generates shuffled single precision floating point exponential deviates ((\lambda=1)).</td>
</tr>
</tbody>
</table>

**TABLE 1.**

ENTRY POINT NAMES OF CONTROL SECTION OVFLOW
IX is the starting value of the sequence and may contain any integer number between 1 and 2147483647. This variable should not be altered by the user during the execution of the program, unless it is desired to repeat a sequence of random numbers.

A is either a scalar or vector variable and is the location with a specified dimension into which the random number or numbers are stored (see next parameter). Note that for entry points INT and SINT, this argument should be of INTEGER type.

N is an integer variable or constant designating how many random numbers are to be generated during this call. If N is greater than 1, A above must be a vector dimensioned at least as large as N. If N is equal to 1, then A may be scalar.

Some sample programs are given below:

(1) To generate 1000 consecutive integer random numbers:

```fortran
INTEGER*4 M(1000)
CALL OVFLOW
IX = 1234567
CALL INT (IX, M, 1000)
END
```

(2) To generate 25 shuffled single precision floating point normal deviates and scale to mean 10 and standard deviation 5:

```fortran
REAL*4 A(25)
CALL OVFLOW
JJ = 1936748
N = 25
```
CALL SNORM (JJ, A, N)
DO 1 I = 1, 25
A(I) = A(I)*5.0 + 10.0
1 CONTINUE

--
--
--

END

(3) To generate one single precision floating point exponential
deviate with mean 6:
CALL OVFLOW
I9 = 98367221
--
--
--

CALL EXPON (I9, E, 1)
E = E*6.0
--
--
--

END

A. Implementation. LLRANDOM was designed and coded to run under Operating
System/360 (OS). The Assembler Language control section contains a SPIE
(Set Program Interrupt Exit) macro instruction which is a part of the OS
Supervisor Services. This macro enables LLRANDOM to correct for the
fixed point overflows resulting from the division simulation algorithm.
The remainder of the assembly coding is in Basic Assembler Language (BAL), i.e. no other macro calls or supervisor calls. To run LLRANDOM under another operating system for the System/360, an appropriate substitution for the SPIE macro would be necessary.

As currently programmed, LLRANDOM has the following memory requirements:

<table>
<thead>
<tr>
<th>MODULE</th>
<th>SIZE IN BYTES (DECIMAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler CSECT</td>
<td>3571</td>
</tr>
<tr>
<td>FORTRAN function R NORTH</td>
<td>1512</td>
</tr>
<tr>
<td>FORTRAN function R EXP</td>
<td>1106</td>
</tr>
<tr>
<td>Total memory requirement</td>
<td>6189</td>
</tr>
</tbody>
</table>

The System/360 internal timer is rather crude for timing the execution of programs. The following times are therefore approximate timings for the generation of pseudo-random numbers on a System/360 Model 67-2.

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>TIME IN MICROSECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>10.7</td>
</tr>
<tr>
<td>SINT</td>
<td>15.7</td>
</tr>
<tr>
<td>RANDOM</td>
<td>15.6</td>
</tr>
<tr>
<td>SRAND</td>
<td>20.0</td>
</tr>
<tr>
<td>NORMAL</td>
<td>57.5</td>
</tr>
<tr>
<td>SNORM</td>
<td>65.8</td>
</tr>
<tr>
<td>EXPON</td>
<td>59.1</td>
</tr>
<tr>
<td>SEXPON</td>
<td>68.4</td>
</tr>
</tbody>
</table>

(Polar method takes 349 microseconds)  
(Logarithm method takes 132 microseconds)
B. **Future Enhancements.** The normal and exponential deviate routines in LLRANDOM are patterned after a package, SUPER-DUPER, available from Professor G. Marsaglia at McGill University in Montreal. It is available at the Naval Postgraduate School. Marsaglia uses a different multiplier, A, and modulus, m, in his congruential generator from that used in LLRANDOM and he then exclusive OR's this result with the output of a feedback shift register generator. SUPER-DUPER provides only one deviate per call and does not provide for shuffling the sequence.

The two FORTRAN function subprograms, RNORTH and REXPTH, are taken (with slight modification) directly from SUPER-DUPER. Among the changes to be made to LLRANDOM will be to rewrite RNORTH and REXPTH in System/360 Assembly Language and incorporate them directly into the package.

We have experienced occasions where large-scale simulations have been coded in FORTRAN using double precision variables. The fact that LLRANDOM returns single precision numbers causes some inconvenience. To alleviate this problem we will provide the capability in LLRANDOM to return **single precision** numbers into double precision variables or arrays. Note that the values returned will still be single precision; however, they will be stored properly into double precision locations.

Finally, additional entry points will be added to provide **single precision floating point** gamma deviates. Shuffling of the gamma deviates will also be available.

Other enhancements are under consideration.
REFERENCES


**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM ****

C
REXP. TOOTH FUNCTION
FUNCTION REXPTH(K,IX)
DIMENSION C(165)
DATA C/Z40F00000,Z40E10000,Z40D40000,Z40C70000,Z40BB0000,
$ Z40AF0000,Z409B0000,Z40910000,Z40890000,Z40880000,
$ Z407B0000,Z406A0000,Z40640000,Z405E0000,Z40580000,
$ Z40530000,Z404E0000,Z40490000,Z40440000,Z40400000,
$ Z403F0000,Z40350000,Z40320000,Z402F0000,Z402C0000,
$ Z40270000,Z40240000,Z40220000,Z40200000,Z401E0000,
$ Z401A0000,Z40190000,Z40170000,Z40160000,Z40150000,
$ Z40120000,Z40110000,Z40100000,Z3F000000,Z3F000000,
$ Z3F600000,Z3F600000,Z3F700000,Z3F700000,Z3F600000,
$ Z3F650000,Z3F400000,Z3F400000,Z3F400000,Z3F400000/
DATA II/ZFB4FA91/
IF(K.GT.1)GO TO 5
1 CALL RANDOM(Ix,UL,1)
IF(UL.GT.7917049) GO TO 3
T=1-1.2359624 Ul
REXPTH=-ALOG(T)
J=16.*REXPTH+1.
CALL RANDOM(Ix,Z9,1)
IF(Z9*(.0604*T-.0039).GT.T-C(J)) GO TO 1
RETURN
3 REXPTh=19.20352*UL-1.20352
J=16.*REXPTh+1.
EX=EXP(-REXPTh)
CALL RANDOM(Ix,Z9,1)
IF(Z9*(.0604*EX+.0039).GT.EX-C(J)) GO TO 1
RETURN
5 CALL RANDOM(Ix,UL,1)
IF(UL.EQ.01)GO TO 5
REXPTh=4.*-ALOG(UL)
RETURN
END

REXP0010
REXP0020
REXP0030
REXP0040
REXP0050
REXP0060
REXP0070
REXP0080
REXP0090
REXP0100
REXP0110
REXP0120
REXP0130
REXP0140
REXP0150
REXP0160
REXP0170
REXP0180
REXP0190
REXP0200
REXP0210
REXP0250
REXP0260
REXP0270
REXP0280
REXP0290
REXP0300
REXP0310
REXP0320
REXP0330
REXP0340
REXP0350
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM ****

C RNCR TOOTH FUNCTION

FUNCTION RNORTH(K,IX)
  DIMENSION C(45)
  DATA C/240FD25F,240FD25F,240FA9A0,240F5A648,240F32496,
  240E2131,240E6C1A,240E19B5,240DA13E9,240D28E87,240C887BE,
  240C102A4,24086FDD,240ACF513,240A2EE4A,2409BE7BC,240916269,
  240875AC,2407B54D6,24073AEDD,24068CBF6,24061C22C,2405A3D15,
  24052B7FE,2404832E7,2403C2D9D,24032C8B9,240375554,2402FA03D,
  2402A9CDB,240259973,24020980E,2401E145C,2401910F7,240168F45,
  24014D93,240118B20,23FFOA2E4,23FC887BE,23FA06C98,23F785172,
  23F785172,23F50364C,23F50364C,23F50364C/
  DATA 11/2FBC35400/,12/ZFE79702E/
  IF(K.GT.11)GO TO 3
  CALL RANDOM(IY,9,1)
  CALL RANDOMC(IY,T,1)
  B=AINT(7.*(S+T)+37.*ABS(S-T))
  CALL RANDOMC(IY,Z9,1)
  CALL RANDOMC(IY,Z8,1)
  X=9-Z8
  RNORTH=.0625*(X+SIGN(B,X))
RETURN
3  IF(K.GT.12)GO TO 5
4  CALL RANDOMC(IY,Z9,1)
5  CALL RANDOMC(IY,Z8,1)
  IF(Z8.GT.5.50) Z9E=Z9
  RNORTH=2.75*Z9
  J=16.*ABS(RNORTH)+1.
  IF(J-14) 6,6,7
6  P=(J+J-1).#1497466E-2
7  GO TO 8
8  CALL RANDOMC(IY,Z9,1)
  IF(Z9.GT.79.78817E-3
  IF(CJ.GT.79.788466*EXP(-.5*RNORTH)*RNORTH)
  S=-CJ-P*(J-16.*ABS(RNORTH))) GOTO4
RETURN
5  CALL RANDOMC(IY,V,1)
  CALL RANDOMC(IY,Z9,1)
  IF(Z9.GT.0.5) V=E-1
  IF(V.EQ.0) GO TO 5
  X=SQR(7.5625-2.*ALOG(ABS(V)))
  CALL RANDOMC(IY,Z9,1)
  IF(Z9*X.GT.7.275) GO TO 5
  RNORTH=SIGN(X,V)
RETURN
END
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

OVFLOW
CSECT
EXITN RNORTH, REXPTh
ENTRY INT, INT, RANDOM, SRA, SD, SNORM, SEXP, SEXP
USING OVFLOW, R12
B _12(R15) BRANCH AROUND ID
DC AL144
DC CL6ºOVFLOW
STM R14, R12, 12(R13) SAVE REGISTERS IN HIGH SAVE AREA
LR R12, R15 ESTABLISH BASE ADDRESS
ST R13, SA+4 SAVE CALLER'S R13
LR R2, R13
LA R13, SA NEW SAVE AREA
ST R13, 8(R2) STORE WITH CALLING ROUTINE

* ISSUE SPIE TO GET FIXED POINT OVERFLOWS AS WELL AS FORTRAN
* INTERRUPTS.

SPIE FIXIT, (8, 9, 12, 13, 15)
ST R1, PICA SAVE FORTRAN'S PICA ADDRESS
L R13, SA+4 RESTORE CALLER'S R13
LM R14, R12, 12(R13) RESTORE THE REGISTERS
BCR 15, R14 RETURN

LLRA0010
LLRA0020
LLRA0030
LLRA0040
LLRA0050
LLRA0060
LLRA0070
LLRA0080
LLRA0090
LLRA0100
LLRA0110
LLRA0120
LLRA0130
LLRA0140
LLRA0150
LLRA0160
LLRA0170
LLRA0180
LLRA0190
LLRA0200
LLRA0210
LLRA0220
**SPIE BRINGS US HERE ON INTERRUPTS**

**FIXIT**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TM 7(R1),X'F7'</code></td>
<td>Using *R15, was it a fixed point overflow?</td>
</tr>
<tr>
<td><code>BC 5,FORT</code></td>
<td>No, let Fortran's SPIE handle it</td>
</tr>
<tr>
<td><code>CLS 17(3,R1),AINT+1</code></td>
<td>Test whether base of interrupted routine was between entries INT and ASEXPO inclusive; if not, ignore</td>
</tr>
<tr>
<td><code>BL 01,R14</code></td>
<td>The interrupt</td>
</tr>
<tr>
<td><code>CLS 17(3,R1),ASEXPO+1</code></td>
<td>Add 2<strong>31-3 to make 2</strong>31+1 correction</td>
</tr>
<tr>
<td><code>A R4,PM2</code></td>
<td>Add 4 more to make 2**31+1 correction</td>
</tr>
<tr>
<td><code>AR R4,R2</code></td>
<td>All fixed, continue</td>
</tr>
<tr>
<td><code>BR R14</code></td>
<td>Not fixed point overflow, let Fortran</td>
</tr>
<tr>
<td><code>L R15,P1A</code></td>
<td>Extended error handling routine</td>
</tr>
<tr>
<td><code>FR R15</code></td>
<td>Take care of it</td>
</tr>
</tbody>
</table>
**NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM****

**ENTRY POINT : INT**

**BASE REGISTER**

**BRANCH AROUND ID**

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>USING</th>
<th>CLOC</th>
<th>DLOC</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>INT</td>
<td>15</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>R11</td>
<td>4</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>R4</td>
<td>9</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>R2</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>R1</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

**BASE REGISTER**

- R13:SA+4
- R12:SA+4
- R11:SA+4
- R10:SA+4
- R9:SA+4
- R8:SA+4
- R7:SA+4
- R6:SA+4
- R5:SA+4
- R4:SA+4
- R3:SA+4
- R2:SA+4
- R1:SA+4

**BRANCH AROUND ID**

- R13:SA+4
- R12:SA+4
- R11:SA+4
- R10:SA+4
- R9:SA+4
- R8:SA+4
- R7:SA+4
- R6:SA+4
- R5:SA+4
- R4:SA+4
- R3:SA+4
- R2:SA+4
- R1:SA+4

**CONSIDER THE FOLLOWING ARGUMENTS TO FILL THE STORAGE AREA IN HIGH SAVE AREA IN LOW SAVE AREA**

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>USING</th>
<th>CLOC</th>
<th>DLOC</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>INT</td>
<td>15</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>R11</td>
<td>4</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>R4</td>
<td>9</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>R2</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>R1</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
</tbody>
</table>

**BRANCH AROUND ID**

- R13:SA+4
- R12:SA+4
- R11:SA+4
- R10:SA+4
- R9:SA+4
- R8:SA+4
- R7:SA+4
- R6:SA+4
- R5:SA+4
- R4:SA+4
- R3:SA+4
- R2:SA+4
- R1:SA+4

**CONSIDER THE FOLLOWING ARGUMENTS TO FILL THE STORAGE AREA IN HIGH SAVE AREA IN LOW SAVE AREA**

- R13:SA+4
- R12:SA+4
- R11:SA+4
- R10:SA+4
- R9:SA+4
- R8:SA+4
- R7:SA+4
- R6:SA+4
- R5:SA+4
- R4:SA+4
- R3:SA+4
- R2:SA+4
- R1:SA+4

**STORE AS STARTING VALUE FOR NEXT CALL**

- R13:SA+4
- R12:SA+4
- R11:SA+4
- R10:SA+4
- R9:SA+4
- R8:SA+4
- R7:SA+4
- R6:SA+4
- R5:SA+4
- R4:SA+4
- R3:SA+4
- R2:SA+4
- R1:SA+4
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANOM *****

* ENTRY POINT : SINT   LLRA0760
* CNOP 0,8   LLRA0770
USING SINT, R15   BASE REGISTER   LLRA0780
SINT B 10(C,R15)  BRANCH AROUND ID   LLRA0790
DC AL14(4)   LLRA0800
DC CL4*SINT'   LLRA0810
STM R14, R12, 12(R13)  SAVE REGISTERS IN HIGH SAVE AREA   LLRA0820
ST R13, SA+4  ADDRESS OF HIGH SAVE AREA IN LOW SAVE AR.   LLRA0830
LR R2, R13  COPY TO R2   LLRA0840
LA R13, SA  ADDRESS OF LOW SAVE AREA   LLRA0850
ST R13, 8(, R2)  ADDRESS OF LOW SAVE AREA IN HIGH SAVE AR.   LLRA0860
L R9, A75  LOAD MULTIPLIER   LLRA0870
LA R2, 4  CONSTANT FOR BXLE   LLRA0880
LM R5, 7, 0(R1)  ADDRESSES OF THREE ARGUMENTS   LLRA0890
L R5, 0(R5), R5  LOAD STARTING VALUE INTO R5   LLRA0900
L R3, 0(R7)  NUMBER OF CONSECUTIVE WORDS TO FILL   LLRA0910
SLA R3, 2  CONVERT TO BYTES   LLRA0920
SR R0, R2  BACKUP ONE WORD IN CALLER'S ARRAY   LLRA0930
LR R7, R2  INITIAL VALUE FOR INDEX REGISTER   LLRA0940
LA R8, TABLE  ADDRESS OF SHUFFLING TABLE   LLRA0950
L R1, MASK  INDEX MASK FOR SHUFFLING   LLRA0960
CNOP 0, 8  ALIGN BXLE LOOP FOR SPEED   LLRA0970
MR R4, R9  FORM PRODUCT OF A AND X(N-1)   LLRA0980
L2 SLDA R4, 1  R4 = REMAINDER : R5 = QUOTIENT   LLRA1000
SRL R5, 1  ADD QUOTIENT TO REMAINDER THEREBY   LLRA1010
AR R4, R5  SIMULATING DIVISION BY 2**31-1   LLRA1020
LR R5, R4  PUT X(N) INTO R5 FOR NEXT GC AROUND   LLRA1030
NR R4, R1  EXTRACT RIGHT-MOST 7 BITS   LLRA1040
SLA R4, 2  CONVERT TO BYTE OFFSET IN TABLE   LLRA1050
LM R0, 0(R4, R8)  SELECT RANDOM TABLE VALUE   LLRA1060
ST R5, 0(R4, R8)  REPLACE TABLE VALUE WITH X(N)   LLRA1070
ST R0, 0(R7, R6)  RANDOM TABLE VALUE TO CALLER'S ARRAY   LLRA1080
N2 BXLE R7, R2, L2  LOOP AROUND AGAIN   LLRA1090
L R13, SA+4  RESTORE CALLER'S SAVE AREA POINTER   LLRA1100
L R1, 24(, R13)  GET ARGUMENT LIST POINTER AGAIN   LLRA1110
L R4, 0(, R1)  GET STARTING VALUE ADDRESS AGAIN   LLRA1120
ST R5, 0(, R4)  STORE AS STARTING VALUE FOR NEXT CALL   LLRA1130
LM R14, R12, 12(R13)  RESTORE THE REGISTERS   LLRA1140
BCR 15, R14  RETURN   LLRA1150
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

* ENTRY POINT : RANDOM

CNOP 0,8
USING RANDOM,R15 BASE REGISTER
RANDOM
DC AL1(6)
DC C16"RANDOM"
STM R14,R12,12(R13) SAVE REGISTERS IN HIGH SAVE AREA
ST R13,SA+4 ADDRESS OF HIGH SAVE AREA IN LOW SAVE AR.
LR R2,R13 COPY TO R2
LA R13,SA ADDRESS OF LOW SAVE AREA
ST R13,R1,R2 ADDRESS OF LOW SAVE AREA IN HIGH SAVE AR.
LM R9,R11,A75 LOAD MULTIPLIER AND NORMALIZATION CONST.
LA R2,79 CONSTANT FOR BXLE
LM R5,R7,0(R1) ADDRESSES OF THREE ARGUMENTS
L R8,0(R1) LOAD STARTING VALUE INTO R5
LL R3,0(R7) NUMBER OF CONSECUTIVE WORDS TO FILL
SLA R3,2 CONVERT TO BYTES
SR R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY
LR R7,R2 INITIAL VALUE FOR INDEX REGISTER
SUR F0,FR0 CLEAR FLOATING POINT REGISTER 0
LA R12,N3 ADDRESS OF BXLE INSTRUCTION
LA R13,M3 ADDRESS OF NORMALIZATION ROUTINE
CNOP 0,8 ALIGN BXLE LOOP FOR SPEED
L3 MR R4,R9 FORM PRODUCT OF A AND X(N-1)
SLQA R4,1 R4 = REMAINDER /17 = QUOTIENT
SRL R4,7 ADD QUOTIENT TO REMAINDER THEREBY
AR R4,R5 SIMULATING DIVISION BY 2**31-1
LR R5,R4 PUT X(N) INTO R5 FOR NEXT CALL OR AROUND
SRL R4,7 MAKE ROOM FOR THE EXPONENT
OR R4,R10 OR ON THE EXPONENT
ST R4,0(R7,R6) STORE IN CALLER'S ARRAY
CR R4,R11 DID IT NEED NORMALIZATION?
BML 4,R13 YES, GO NORMALIZE IT
L3 BXLE R7,R2,L3 LOOP AROUND AGAIN
L R4,0(R1) GET STARTING VALUE ADDRESS AGAIN
ST R5,0(R4) STORE AS STARTING VALUE FOR NEXT CALL
LM R14,R12,12(R13) RESTORE THE REGISTERS
BML 15,R14 RETURN
M3 LE FR2,0(R7,R6) LOAD INTO FLOATING POINT REGISTER 2
AER FR2,FR0 ADD ZERO AND NORMALIZE
STE FR2,0(R7,R6) STORE BACK NORMALIZED
BR R12 CONTINUE THE BXLE LOOP
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR: LLRANDOM *****

* ENTRY POINT: SRAND

** USING SRAND,R15 BASE REGISTER **

B 101(R15) BRANCH AROUND ID

DC AL15(5)  \[LLRA1680\]

DC CLS*SRAND* \[LLRA1690\]

STM R14,R12,R17(R13) SAVE REGISTERS IN HIGH SAVE AREA \[LLRA1790\]

LR R2,R13 COPY TO R2 \[LLRA1740\]

LA R15,SAX \[LLRA1750\]

ST R13,SA+4 ADDRESS OF HIGH SAVE AREA IN LOW SAVE AREA \[LLRA1770\]

LR R2,R13 COPY TO R2 \[LLRA1740\]

LA R15,SAX \[LLRA1750\]

ST R13,SA+4 ADDRESS OF LOW SAVE AREA IN HIGH SAVE AREA \[LLRA1770\]

LM R9,R11,A75 LOAD MULTIPLIER AND NORMALIZATION CONST. \[LLRA1790\]

LA R2,4 CONSTANT FOR BXLE \[LLRA1780\]

LM R5,R7,0(R1) ADDRESSES OF THREE ARGUMENTS \[LLRA1790\]

L R5,0(R5) LOAD STARTING VALUE INTO R5 \[LLRA1800\]

L R3,0(R7) NUMBER OF CONSECUTIVE WORDS TO FILL \[LLRA1810\]

SLA R3,2 CONVERT TO BYTES \[LLRA1820\]

SR R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY \[LLRA1830\]

LR R7,R2 INIT VALUE FOR INDEX REGISTER \[LLRA1840\]

SUR FR0,FR0 CLEAR FLOATING POINT REGISTER 0 \[LLRA1850\]

LA R12,N4 ADDRESS OF BXLE INSTRUCTION \[LLRA1860\]

LA R13,M4 ADDRESS OF NORMALIZATION ROUTINE \[LLRA1870\]

LA R8,Table ADDRESS OF SHUFFLING TABLE \[LLRA1880\]

L R1,MASK INDEX MASK FOR SHUFFLING \[LLRA1890\]

CNOP 0,8 ALIGN BXLE LOOP FOR SPEED \[LLRA1900\]

L R4,R9 FORM PRODUCT OF A AND X(N-1) \[LLRA1910\]

SLDA R4,1 R4 = REMAINDER \[LLRA1920\]

SRL R5,1 ADD QUOTIENT TO REMAINDER THEREBY \[LLRA1930\]

AR R4,R5 SIMULATING DIVISION BY 2**31-1 \[LLRA1940\]

LR R5,R4 PUT X(N) INTO R5 FOR NEXT GC AROUND \[LLRA1950\]

NR R4,R1 EXTRACT RIGHT-MOST 7 BITS \[LLRA1960\]

SRA R4,2 CONVERT TO BYTE OFFSET IN TABLE \[LLRA1970\]

LE R0,0(R4,R8) SELECT RANDOM TABLE VALUE \[LLRA1980\]

STM R5,0(R4,R8) REPLACE TABLE VALUE WITH X(N) \[LLRA1990\]

SLA R0,7 MAKE ROOM FOR THE EXPONENT \[LLRA2000\]

OR R0,R10 \[LLRA2010\]

ST R0,0(R7,R6) STORE IN CALLER'S ARRAY \[LLRA2020\]

CR R0,R11 DID IT NEED NORMALIZATION ? \[LLRA2030\]

BCR 4,R13 YES, GO NORMALIZE IT \[LLRA2040\]

BXLE R7,R2,L4 LOCP AROUND AGAIN \[LLRA2050\]

L R13,SA+4 RESTORE CALLER'S SAVE AREA POINTER \[LLRA2060\]

L R1,241(R13) GET ARGUMENT LIST POINTER AGAIN \[LLRA2070\]

LM R4,0(R1) GET STARTING VALUE ADDRESS AGAIN \[LLRA2080\]

ST R5,0(R4) STORE AS STARTING VALUE FOR NEXT CALL \[LLRA2090\]

LM R14,R12,12(R13) RESTORE THE REGISTERS \[LLRA2100\]

BCR 15,R14 RETURN \[LLRA2110\]

LE FR2,0(R7,R6) LOAD INTO FLOATING POINT REGISTER 2 \[LLRA2120\]
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM ****

AER   FR2,FR0   ADD ZERO AND NORMALIZE   LLRA2130
STE   FR2,CR7,R61 STORE BACK NORMALIZED   LLRA2140
BR    R12      CONTINUE THE BXLE LOOP    LLRA2150
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

* ENTRY POINT : NORMAL

CNOP 0,8

USING NORMAL,R15 BASE REGISTER

NORMAL

B 12(R15) BRANCH AROUND 1D

DC AL1(6)

DC CL6'NORMAL'

STM R14,R12,R12(R13) SAVE REGISTERS IN HIGH SAVE AREA

ST R13,SA24+4 ADDRESS OF HIGH SAVE AREA IN LOW SAVE AR.

LR R2,R13 COPY TO R2

LA R13,SA2 ADDRESS OF LOW SAVE AREA

ST R13,R2(R2) ADDRESS OF LOW SAVE AREA IN HIGH SAVE AR.

LM R9,R11,A75N LOAD MULTIPLIER,EXponent, AND TEST MASK

LA R2,4 CONSTANT FOR BXLE

LM R5,R7,0(R1) ADDRESSES OF THREE ARGUMENTS

L R5,0(R1) LOAD STARTING VALUE INTO R5

L R3,0(R7) NUMBER OF CONSECUTIVE WORDS TO FILL

S LA R3,2 CONVERT TO BYTES

SR R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY

LR R7,R2 INITIAL VALUE FOR INDEX REGISTER

LA R13,TABLE ADDRESS OF TABLE OF CONSTANTS

LA R12,N5 ADDRESS OF BXLE

C NOP 0,8 ALIGN BXLE LOOP FOR SPEED

MR R4,R9 FORM PRODUCT OF A AND X(N-1)

SLDA R4,1 R4 = REMAINDER : R5 = QUOTIENT

SRL R5,1 ADD QUOTIENT TO REMAINDER THEREBY

AR R4,R5 SIMULATING DIVISION BY 2**31-1

LR K5,R4 PUT X(N) INTO R5

LR R0,R5 COPY R5 INTO R0 FOR NOW

NR R5,R11 SHOULD WE MAKE IT NEGATIVE ?

BC B,F,1 POSITIVE, KEEP GOING

LNR R5,R5 MAKE R5 TRUE NEGATIVE

SLR R4,R4 CLEAR R4 TO ZERO

FL CL R5,C1 R5 LESS THAN X*68000000* ?

BC 11,F2 NO

SLDL R4,8 SHIFT FIRST 8 BITS OF R5 INTO R4 AS INDEX

IC R4,0(R4,R13) OBTAIN CONSTANT FROM TABLE

STC R4,PWR+1 STORE IN SECOND BYTE OF PWRD

SRL R5,8 SHIFT REMAINING 24 BITS RIGHT THEN OR ON

ALR R5,R10 EXPONENT TO MAKE +24 BITS/16

ST R5,0(R7,R6) STORE IN CALLER'S ARRAY

LE FRO,PWRD LOAD CHARACTERISTIC TO FLOATING POINT

AE FRO,0(R7,R6) REGISTER 0 AND ADD FRACTION

STE FRO,0(R7,R6) STORE NORMAL DEVIATE IN CALLER'S ARRAY

LR R9,R0 COPY BACK TO R5 FOR NEXT CC AROUND

BR R12 GO TO BXLE AND CONTINUE

F2 CL R5,C2 R5 LESS THAN X*60000000* ?

BC 11,F3 NO
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLDL R4,8</td>
<td>Shift first 8 bits of R5 into R4 as index</td>
<td>LLRA2660</td>
</tr>
<tr>
<td>SL R4,C3M</td>
<td>Subtract E8</td>
<td>LLRA2670</td>
</tr>
<tr>
<td>IC R4,0(R4,R13)</td>
<td>Obtain constant from table</td>
<td>LLRA2680</td>
</tr>
<tr>
<td>STC R4,NWDR+1</td>
<td>Store in second byte of NWDR</td>
<td>LLRA2690</td>
</tr>
<tr>
<td>SRL R5,8</td>
<td>Shift remaining 24 bits right then or on</td>
<td>LLRA2700</td>
</tr>
<tr>
<td>ALR R5,R10</td>
<td>Exponent to make *(24 bits)/16</td>
<td>LLRA2710</td>
</tr>
<tr>
<td>ST R5,0(R7,R6)</td>
<td>Store in caller's array</td>
<td>LLRA2720</td>
</tr>
<tr>
<td>LE FR0,NWDR</td>
<td>Load characteristic to floating point</td>
<td>LLRA2730</td>
</tr>
<tr>
<td>SE FR0+0(R7,R6)</td>
<td>Register 0 and subtract fraction</td>
<td>LLRA2740</td>
</tr>
<tr>
<td>STE FR0+0(R7,R6)</td>
<td>Store normal deviate in caller's array</td>
<td>LLRA2750</td>
</tr>
<tr>
<td>LR R5,R0</td>
<td>Copy back to R5 for next GC around</td>
<td>LLRA2760</td>
</tr>
<tr>
<td>BR R12</td>
<td>Go to BXLE and continue</td>
<td>LLRA2770</td>
</tr>
<tr>
<td>CL R5,C3</td>
<td>R5 less than X*E2F00000?</td>
<td>LLRA2780</td>
</tr>
<tr>
<td>BC 11,F4</td>
<td>No</td>
<td>LLRA2790</td>
</tr>
<tr>
<td>SLDL R4,12</td>
<td>Shift first 12 bits of R5 into R4</td>
<td>LLRA2800</td>
</tr>
<tr>
<td>SL R4,C2M</td>
<td>Subtract C8</td>
<td>LLRA2810</td>
</tr>
<tr>
<td>IC R4,0(R4,R13)</td>
<td>Obtain constant from table</td>
<td>LLRA2820</td>
</tr>
<tr>
<td>STC R4,PWDR+1</td>
<td>Store in second byte of PWDR</td>
<td>LLRA2830</td>
</tr>
<tr>
<td>SRL R5,8</td>
<td>Shift remaining 20 bits right then or on</td>
<td>LLRA2840</td>
</tr>
<tr>
<td>ALR R5,R10</td>
<td>Exponent to make *(20 bits)/16</td>
<td>LLRA2850</td>
</tr>
<tr>
<td>ST R5,0(R7,R6)</td>
<td>Store in caller's array</td>
<td>LLRA2860</td>
</tr>
<tr>
<td>LE FR0,PWDR</td>
<td>Load characteristic to floating point</td>
<td>LLRA2870</td>
</tr>
<tr>
<td>AE FR0,0(R7,R6)</td>
<td>Register 0 and add fraction</td>
<td>LLRA2880</td>
</tr>
<tr>
<td>STE FR0,0(R7,R6)</td>
<td>Store normal deviate in caller's array</td>
<td>LLRA2890</td>
</tr>
<tr>
<td>LR R5,R0</td>
<td>Copy back to R5 for next GC around</td>
<td>LLRA2900</td>
</tr>
<tr>
<td>BR R12</td>
<td>Go to BXLE and continue</td>
<td>LLRA2910</td>
</tr>
<tr>
<td>CL R5,C4</td>
<td>R5 less than X*E2E00000?</td>
<td>LLRA2920</td>
</tr>
<tr>
<td>BC 11,F5</td>
<td>No</td>
<td>LLRA2930</td>
</tr>
<tr>
<td>SLDL R4,12</td>
<td>Shift first 12 bits of R5 into R4</td>
<td>LLRA2940</td>
</tr>
<tr>
<td>SL R4,C3M</td>
<td>Subtract E17</td>
<td>LLRA2950</td>
</tr>
<tr>
<td>IC R4,0(R4,R13)</td>
<td>Obtain constant from table</td>
<td>LLRA2960</td>
</tr>
<tr>
<td>STC R4,NWDR+1</td>
<td>Store as second byte of NWDR</td>
<td>LLRA2970</td>
</tr>
<tr>
<td>SRL R5,8</td>
<td>Shift remaining 20 bits right then or on</td>
<td>LLRA2980</td>
</tr>
<tr>
<td>ALR R5,R10</td>
<td>Exponent to make *(20 bits)/16</td>
<td>LLRA2990</td>
</tr>
<tr>
<td>ST R5,0(R7,R6)</td>
<td>Store in caller's array</td>
<td>LLRA3000</td>
</tr>
<tr>
<td>LE FR0,NWDR</td>
<td>Load characteristic to floating point</td>
<td>LLRA3010</td>
</tr>
<tr>
<td>SE FR0,0(R7,R6)</td>
<td>Register 0 and subtract fraction</td>
<td>LLRA3020</td>
</tr>
<tr>
<td>STE FR0,0(R7,R6)</td>
<td>Store normal deviate in caller's array</td>
<td>LLRA3030</td>
</tr>
<tr>
<td>LR R5,R0</td>
<td>Copy back to R5 for next GC around</td>
<td>LLRA3040</td>
</tr>
<tr>
<td>BR R12</td>
<td>Go to BXLE and continue</td>
<td>LLRA3050</td>
</tr>
<tr>
<td>ST R5,FWDR</td>
<td>Store R5 in argument list</td>
<td>LLRA3060</td>
</tr>
<tr>
<td>ST R0,XWDR</td>
<td>Pass starting value</td>
<td>LLRA3070</td>
</tr>
<tr>
<td>LA R13,S2A</td>
<td>Load low save area pointer</td>
<td>LLRA3080</td>
</tr>
<tr>
<td>LA R1,FLIST</td>
<td>Argument list for call Te RNORTH</td>
<td>LLRA3090</td>
</tr>
<tr>
<td>LR R8,R15</td>
<td>Copy base register for BALR linkage</td>
<td>LLRA3100</td>
</tr>
<tr>
<td>LF R15,AHOR</td>
<td>Address of function subroutine RNORTH</td>
<td>LLRA3110</td>
</tr>
<tr>
<td>BALR R14,R15</td>
<td>Branch to RNORTH</td>
<td>LLRA3120</td>
</tr>
<tr>
<td>LR R15,R8</td>
<td>Restore base register</td>
<td>LLRA3130</td>
</tr>
<tr>
<td>STE FR0,0(R7,R6)</td>
<td>Store normal deviate in caller's array</td>
<td>LLRA3140</td>
</tr>
</tbody>
</table>
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

L   R5,XWRD    NEW STARTING VALUE    LLRA3150
LA  R13,ATBLE  RESTORE R13 TO TABLE OF CONSTANTS  LLRA3160
N5  BXLE R7,R2,L5   LOOP AROUND AGAIN    LLRA3170
L   R13,SA2+4   RESTORE HIGH SAVE AREA POINTER  LLRA3180
L   R1,241,(13) GET ARGUMENT LIST POINTER AGAIN  LLRA3190
L   R4,0(1,R1)  GET STARTING VALUE ADDRESS AGAIN  LLRA3200
ST  R5,0(1,R4)  STORE AS STARTING VALUE FOR NEXT CALL  LLRA3210
LM  R14,R12,12(R13) RESTORE THE REGISTERS  LLRA3220
BCR 15,R14   RETURN    LLRA3230
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR: LLRANDOM ****

* ENTRY POINT: SNORM

** CNOP 0,8

** USING SNORM, R15

** SNORM

B 10(R15) BRANCH AROUND 10

DC A11(5)  

DC CL5*SNORM'

SIM F14,R12,12(R13) SAVE REGISTERS IN HIGH SAVE AREA

ST R13,SA2+4 ADDRESS OF HIGH SAVE AREA IN LOW SAVE AREA

LA R13,SA2 ADDRESS OF LOW SAVE AREA

ST R13,8(R2) ADDRESS OF LOW SAVE AREA IN HIGH SAVE AREA

LM R9, R11, A75N LOAD MULTIPLIER, EXPONENT, AND TEST MASK

LA R2, 4 CONSTANT FOR BXCING

LM R5,R7,0(R1) ADDRESSES OF THREE ARGUMENTS

L R5,01(R5) LOAD STARTING VALUE INTO R5

LL R3,01(R7) NUMBER OF CONSEQUENTIAL WORDS TO FILL

SLA R3,2 CONVERT TO BYTES

SR R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY

LR R7,R2 INITIAL VALUE FOR INDEX REGISTER

LA R13,ATABLE ADDRESS OF TABLE OF CONSTANTS

LA R8, TABLE ADDRESS OF SHUFFLING TABLE

LA R12, N6 ADDRESS OF BXL

L R1,MASK INDEX MASK FOR SHUFFLING

** CNOP 0,8 ALIGN BXLE LOOP FCR SPEED

L6 MR R4,R9 FORM PRODUCT OF A AND X(N-1)

SLDA R4,1 R4 = REMAINDER; R5 = QUOTIENT

SRL R5,1 ADD QUOTIENT TO REMAINDER THEREBY

AR R4,R5 SIMULATING DIVISION BY 2**31-1

LR R5,R4 PUT X(N) INTO R5

NR R4,R1 EXTRACT RIGHT-MOST 7 BITS

SLA R4,2 CONVERT TO BYTE OFFSET IN TABLE

L F0,0(R4,R3) SELECT RANDOM TABLE VALUE

ST R5,01(R4,R3) REPLACE TABLE VALUE WITH X(N)

XR R0,R5 EXCHANGE RO AND R5

XR R0,R5 BY EXCLUSIVE ORING

XR R0,R5 THEM WITH EACH OTHER

NR R4,R1 SHOULD WE MAKE IT NEGATIVE?

BC B,fl5 POSITIVE

LNR R5,R5 MAKE R5 TRUE NEGATIVE

SLR R4,R4 CLEAR R4 TO ZERO

CL R5,R1 R5 LESS THAN X*68000000*? 

** F15

SLDL R4,8 SHIFT FIRST 8 BITS OF R5 INTO R4 AS INDEX

IC R4,0(R4,R13) OBTAIN CONSTANT FROM TABLE

STC R4,PWR0+1 STORE IN SECOND BYTE OF PWRD

SRL R5,8 SHIFT REMAINING 24 BITS RIGHT THEN OR ON

ALR R5,R10 EXPONENT TO MAKE *(24 BITS)/16

LRA3250

LRA3260

LRA3270

LRA3280

LRA3290

LRA3300

LRA3310

LRA3320

LRA3330

LRA3340

LRA3350

LRA3360

LRA3370

LRA3380

LRA3390

LRA3400

LRA3410

LRA3420

LRA3430

LRA3440

LRA3450

LRA3460

LRA3470

LRA3480

LRA3490

LRA3500

LRA3510

LRA3520

LRA3530

LRA3540

LRA3550

LRA3560

LRA3570

LRA3580

LRA3590

LRA3600

LRA3610

LRA3620

LRA3630

LRA3640

LRA3650

LRA3660

LRA3670

LRA3680

LRA3690

LRA3700

LRA3710

LRA3720

LRA3730
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

ST  R0,XWRD  PASS STARTING VALUE  LLRA4230
LA  R13,SA2  LOAD LOW SAVE AREA POINTER  LLRA4240
LA  R1,FLIST  ARGUMENT LIST FOR CALL TO RNDRTH  LLRA4250
LR  R8,R15  COPY BASE FOR BALR LINKAGE  LLRA4260
L  R15,ARNR  ADDRESS OF FUNCTION SUB Cutter RNDRTH  LLRA4270
BALR  R14,R15  BRANCH TO RNDRTH  LLRA4280
LR  R15,R5  RESTORE BASE REGISTER  LLRA4290
ST  F13,(R6)  STORE NORMAL DEVIATE IN CALLER'S ARRAY  LLRA4300
L  R5,XWRD  NEW STARTING VALUE  LLRA4310
LA  R13,ATBLE  RESTORE R13 TO TABLE OF CONSTANTS  LLRA4320
LA  R8,TA2LE  RESTORE R8 TO ADDRESS OF SHuffling TABLE  LLRA4330
L  R4,MASK  RESTORE R4 TO INDEX MASK  LLRA4340
BXLE  R7,R2,L6  LOOP AROUND AGAIN  LLRA4350
L  R13,SA2+4  RESTORE HIGH SAVE AREA POINTER  LLRA4360
L  R1,S(R13)  GET ARGUMENT LIST POINTER AGAIN  LLRA4370
L  R4,S(R1)  GET STARTING VALUE ADDRESS AGAIN  LLRA4380
ST  R5,O1,R4  STORE AS STARTING VALUE FOR NEXT CALL  LLRA4390
LM  R14,R12,12(R13)  RESTORE THE REGISTERS  LLRA4400
BCK  13,R14  RETURN  LLRA4410
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR: L&RANDOM ****

* ENTRY POINT : EXPO

CNOP 0,8

USING EXPO, R15

EXPO 101(R15)

BRA 101(R15)

DC AL1(S)

DC CL5*EXPO'

STM R14,R12,R13(R13) SAVE REGISTERS IN HIGH SAVE AREA

STM R13,S12+4 ADDRESS OF HIGH SAVE AREA IN LOW SAVE AREA

LR R2,R13 COPY TO R2

LA R13,S1A2 ADDRESS OF LOW SAVE AREA

STM R9,R11,A75N LCAD MULTIPLIER, EXPONENT, ASC TEST MASK

LML R2,4 CONSTANT FOR BXLE

LML R7,R8,0(R1) ADDRESSES OF THREE ARGUMENTS

L R5,0(R5) LOAD STARTING VALUE INTO R5

L L3,0(R7) NUMBER OF CONSECUTIVE WORDS TO FILL

SR R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY

LR R7,R6 INITIAL VALUE FOR INDEX REGISTER

LA R13,BTBLE ADDRESS OF TABLE OF CONSTANTS

LA R12,R7 ADDRESS OF FOR SIMULATING DIVISION BY 2**31-1

CNOP 0,8 ALIGN BXLE LOOP FOR SPEED

MR R4,R9 FORM PRODUCT OF A AND X(N-1)

SLDA R4,1 R4 = REMAINDER = R5 = QUOTIENT THEREBY

SR R5,1 ADD QUOTIENT TO REMAINDER THEREBY

AR R4,R5 PRODUCTION OF A AND X(N-1)

LR R5,R4 SIMULATING DIVISION BY 2**31-1

LR R0,R5 PUT X(N) INTO R5 FOR NOW

NR R4,R11 SHOULD WE MAKE IT NEGATIVE?

BC 8,E1 POSITIVE, KEEP GOING

LNR R5,R5 MAKE R5 TRUE NEGATIVE

SLR R4,R4 CLEAR R4 TO ZERO

E1 CL R5,D1 R5 LESS THAN X*5000000*?

BC 11,E2 NO

SLCL R4,8 SHIFT FIRST 8 BITS OF R5 INTO R4 AS INDEX

IC R4,0(R4,R13) OBTAIN CONSTANT FROM TABLE

STC R4,PWRD+1 STORE IN SECOND BYTE OF PWRD

SER R5,8 SHIFT REMAINING 24 BITS RIGHT THEN OR ON

ALR R5,R10 EXPONENT TO MAKE .4**31-1/16

ST R5,0(R7,R6) STORE IN CALLER'S ARRAY

LE R0,PWRD LCAD CHARACTERISTIC TO FLOATING POINT

AE R0,0(R7,R6) REGISTER 0 AND ADD EXCLUSIVE

STE R0,CUR(R7,R6) STORE EXCLUSIVE DEVIATE IN ARRAY

LR R5,R0 COPY BACK TO R5 FOR NEXT CC AROUND

BR R12 GO TO BXLE AND CONTINUE

E2 CL R5,D2 R5 LESS THAN X*1700000*?

BC 11,E3 NO
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR: LLRANOD ****

SLDL R4,12
SL R4,01M
IC R4,(R4,R13)
STC R5,01(PWRD)+1
SRL R5,8
ALR R5,R10
ST R5,(R7,R6)
LE FRO,PWRD
AE FRO,0(R7,R6)
STE FRO,0(R7,R6)
LR R5,R0
BR R12
E3
ST R5,EWRD
ST R0,XWRO
LA R13,SA2
LA R1,ELIST
LR R8,R15
L R15,AEXP
BALR R14,R15
LR R15,R8
STE FRO,0(R7,R6)
L R5,XWRO
LA R13,BTLE
BXLE R7,R2,R7
L R13,SA2+4
L R12,24,(R13)
L R4,0,(R1)
ST R5,0,(R4)
LM R14,R12,12(R13)
BCR 15,R14
***** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR : LLRANDOM *****

* ENTRY POINT : SEXPON

* CNOP 0,8

USING SEXPON,R15 BASE REGISTER

SEXPON B 12(R15) BRANCH AROUND ID

AL1(6)

CL6*SEXPON*

R1,R12,R13(R13) SAVE REGISTERS IN HIGH SAVE AREA

R13,SAA ADDRESSES OF HIGH SAVE AREA IN LOW SAVE AREA.

R2,R3 COPY TO R2

R13,SAA ADDRESS OF LOW SAVE AREA

R13,SAA ADDRESS OF LOW SAVE AREA IN HIGH SAVE AREA.

R9,R11,A75N LOAD MULTIPLIER,EXPONENT,ACC TEST MASK

R2,4 CONSTANT FOR BXLE

R5,R7,0(R1) ADDRESSES OF THREE ARGUMENTS

R5,01,R5 LOAD STARTING VALUE IN R5

R3,04,R7 NUMBER OF CONSECUTIVE WORDS TO FILL

R3,2 CONVERT TO BYTES

R6,R2 BACKUP ONE WORD IN CALLER'S ARRAY

R7,R2 INITIAL VALUE FOR INDEX REGISTER

R13,BTBL ADDRESS OF TABLE OF CONSTANTS

R8,TABLE ADDRESS OF SHUFFLING TABLE

R12,N8 ADDRESS OF BXLE

R1 MASK INDEX MASK FOR SHUFFLING

CNOP 0,8 ALIGN BXLE LOOP FOR SPEED

R4,R9 FORM PRODUCT OF A AND X(N-1)

SLDA R4,1 R4 = QUOTIENT ; R5 = REMAINDER

SRL R5,1 ADD QUOTIENT TO REMAINDER THEREBY

AR R4,R5 SIMULATING DIVISION BY 2**31-1

LR R5,R4 PUT X(N) INTO R5

NR R4,R1 EXTRACT RIGHT-MOST 7 BITS

SLA R4,2 CONVERT TO BYTE OFFSET IN TABLE

R0,0(R4,R8) SELECT RANDOM TABLE VALUE

R9,0(R4,R8) REPLACE TABLE VALUE WITH X(N)

XR R9,R5 EXCHANGE RO AND R5

XR R0,R5 BY EXCLUSIVE OR'ING

XR R9,R5 THEM WITH EACH OTHER

NR R4,R11 SHOULD WE MAKE IT NEGATIVE?

BC B,E1S POSITIVE; KEEP GOING

LNR R5,R5 MAKE R5 TRUE NEGATIVE

SRL R4,R4 CLEAR R4 TO ZERO

CE R5,D1 R5 LESS THAN *05000000*?

BC 11,E2S NO

SLDL R4,8 SHIFT FIRST 8 BITS OF R5 INTO R4 AS INDEX

IC R4,0(R4,R13) OBTAIN CONSTANT FROM TABLE

STC R4,PRW+1 STORE IN SECOND BYTE OF PRW

SRL R5,8 SHIFT REMAINING 24 BITS RIGHT THEN OR ON

ALR R5,R10 EXPONENT TO MAKE *(24 BITS)/16
**** NAVAL POSTGRADUATE SCHOOL RANDOM NUMBER GENERATOR: LLRANDOM ****

ST R5,0(R7,R6) STORE IN CALLER'S ARRAY LLRA5720
LE FRO,PRD LOAD CHARACTERISTIC TO FLOATING POINT LLRA5730
AE FRO,0(R7,R6) REGISTER 0 AND ADD FRACTION LLRA5740
STE FRO,0(R7,R6) STORE EXPONENTIAL DEVIATE IN ARRAY LLRA5750
LR R5,R0 COPY BACK TO R5 FOR NEXT CC AROUND LLRA5760
BR R12 GO TO BLXLE AND CONTINUE LLRA5770
E2S CL R5,D2 R5 LESS THAN X*17000000+ ? LLRA5780
BC 11,E3S NO LLRA5790
SLD R4,12 SHIFT FIRST 12 BITS OF R5 INTO R4 LLRA5800
SL R4,DM SUBTRACT CFF LLRA5810
TC R4,0(R4,R13) OBTAIN CONSTANT FROM TABLE LLRA5820
STC R4,PWRD+1 STORE AS SECOND BYTE OF PWRD LLRA5830
SRL R5,8 SHIFT REMAINING 20 BITs RIGHT THEN OR ON LLRA5840
ALR R5,R10 EXPONENT TO MAKE 120 BITS/16 LLRA5850
ST R5,0(R7,R6) STORE IN CALLER'S ARRAY LLRA5860
LE FRO,PWRD LOAD CHARACTERISTIC TO FLOATING POINT LLRA5870
AE FRO,0(R7,R6) REGISTER 0 AND ADD FRACTION LLRA5880
STE FRO,0(R7,R6) STORE EXPONENTIAL DEVIATE IN ARRAY LLRA5890
LR R5,R0 COPY BACK TO R5 FOR NEXT CC AROUND LLRA5900
BR R12 GO TO BLXLE AND CONTINUE LLRA5910
E3S ST R5,EWRD STORE R5 IN ARGUMENT LIST LLRA5920
ST R6,EWRD PASS STARTING VALUE LLRA5930
LA R13,S2A LOAD LOW SAVE AREA POINTER LLRA5940
LA R1,EILIST ARGUMENT LIST FOR CALL TO REXPTH LLRA5950
LR R8,R15 COPY BASE FOR BLR LINKAGE LLRA5960
L R15,AREXP ADDRESS OF FUNCTION SUBROUTINE REXPTH LLRA5970
BALR R14,R15 BRANCH TO REXPTH LLRA5980
LR R15,R8(R8) STORE BASE REGISTER LLRA5990
STE FRO,0(R7,R6) STORE EXPONENTIAL DEVIATE IN ARRAY LLRA6000
L R5,EWRD NEW STARTING VALUE LLRA6010
LA R13,BTBL RESTORE R13 TO TABLE OF CONTENTS LLRA6020
LA R8,TABLE RESTORE TO ADDRESS OF SHUFFLING TABLE LLRA6030
L R1,MSK RESTORE R1 TO INDEX MASK LLRA6040
N8 BLXLE R7,R2,L8 LOOP AROUND AGAIN LLRA6050
L R13,S2A+4 RESTORE HIGH SAVE AREA POINTER LLRA6060
L R1,24,(R13) GET ARGUMENT LIST POINTER AGAIN LLRA6070
L R4),(R1) GET STARTING VALUE ADDRESS AGAIN LLRA6080
ST R5,0(R4) STORE AS STARTING VALUE FOR NEXT CALL LLRA6090
LM R14,R12,12(R13) RESTORE THE REGISTERS LLRA6100
BCR 15,R14 RETURN LLRA6110