Technical Report CS74011-R

IMPLEMENTATION OF FORTRAN RANDOM NUMBER GENERATORS ON COMPUTERS WITH ONE'S COMPLEMENT ARITHMETIC†

Richard E. Nance*
and
Claude Overstreet, Jr.**

July 1974

†Some of the results described are related in an earlier report from the Computer Science/Operations Research Center at Southern Methodist University, Dallas, Texas.

*Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

**Department of Computer Science, Bowling Green State University, Bowling Green, Ohio.

INTRODUCTION

A method of generating random numbers on the SRU 1108 using the FORTRAN language is described in a paper by Marsaglia and Bray [1]. This method seeks to produce "the least positive residue modulo 2^{36} " by using the single statement.

```
I = ISIGN(IABS((I*K))+MINO(ISIGN(1,(I*K)),0),(I*K)).
```

In a subsequent paper Grosenbaugh [2] noted that this instruction failed to produce the exact residues; however, the statement

```
I = I*K+(ISIGN(1,I*K)-ISIGN(1,I))/2
```

does give the correct residues.

Marsaglia and Bray note that the difficulty in producing the correct residues on 1's complement machines such as the SRU 1108 stems from the procedure by which the FORTRAN compiler handles signed numbers. In this paper we explore several methods for producing the correct residues on the two most popular series of 1's complement machines--the SRU 1107-1108 and the Control Data Corporation (CDC) 6000 and CYBER group. In working with both the SRU and CDC series, we treat the "one-line generators" in the sense of the authors [1].

Actually, the FORTRAN routines suggested by Marsaglia and Bray do not produce random values in the usual sense, i.e. fractional values distributed U(0,1)[1]; instead, they produce signed integer residues that must be transformed. In this paper we present several methods for producing residues using the FORTRAN language and compare the execution times for these methods. Each method produces an integer satisfying the usual linear congruence relation; however, differences in the generated residues necessitate slight variations in the conversion of the integers to uniform random variates, i.e. values distributed U(0,1).[1] For example, both instructions above produce positive and negative

---

[1]The notation U(0,1) is used to represent the distribution of values uniformly on (0,1).

residues, and some test is required to assure that the negative residues produce positive uniform variates.

METHODS FOR PRODUCING UNIFORM VARIATES
SRU 1107-1108

We compare four methods for random number generation in FORTRAN on the SRU 1107-1108. In each case the value produced is assigned to an element of a one-dimensional array RANUM. For each method below, the variables IX and IA represent the values X_i and a respectively in the congruence relationship

$$X_{i+1} \equiv aX_i \pmod{2^{35}}, \quad i=0,1,2,\ldots \quad .$$

We also note that the FORTRAN V compiler for the SRU machines contains several non-standard features that are used to advantage. Since our investigation is specific to 1's complement machines, we perceive no gain in restricting our comparisons to the standard subset of the language.

Signed Residue Method

This method uses the single statement suggested by Grosenbaugh [2] to produce signed residues, i.e. both negative and positive residues. A test is required to detect the negative values and to transform them to a congruent positive residue modulo 2^{35}.[2] Division by 2^{35} then produces the uniform variate.

```
IX=IX*IA+(ISIGN(1,IX*IA)-ISIGN(1,IX))/2
IY=IX
IF(IX.LT.0)IY=IX+34359738367
RANUM(I)=IY/34359738368.0
```

[2]The generation of "least positive residues modulo 2^{36}" actually refers to the generation of the complete sets of negative and positive residues modulo 2^{35}.

Unsigned (Positive) Residue Method

An obvious shortcoming of the above technique is the necessity for testing to determine the sign of the generated residues. We develop an alternative that eliminates the explicit test by using the MINO function.

```
IX=IX*IA-MINO(0,ISIGN(34359738367,IX*IA))
RANUM(I)=IX/34359738368.0
```

Masked Sign (FLD Function) Method

FORTRAN V on the SRU 1108 enables bit manipulation capability using the intrinsic function, FLD(J,K,E)[3,5.3]. The arguments of the function are:

J = an INTEGER expression with value $0 \leq J \leq 35$,

K = an INTEGER expression with value $1 \leq K \leq 36$, and

E = any INTEGER, REAL, or LOGICAL expression, a Hollerith word, or a typeless function.

The FLD function extracts a field (or bit string) of K bits from the 36 bit string represented by E beginning with the bit J (counted from left to right where the zero bit is the left most bit of E). After the extracted bit string is right justified, the remaining bits are set to zero. Below the FLD function is used to simply mask the sign bit, i.e. the sign bit is always set to zero thus producing a positive value.

```
IX=FLD(1,35,IX*IA)
RANUM(I)=IX/34359738368.0
```

Logical Mask Method

The fourth method utilizes the logical conjunction operation, which is possible using the AND function in FORTRAN V. By applying the AND function with $2^{35} - 1$ (34359738367) as the second argument, we create a mask to assure that the product is the correct positive residue modulo 2^{35}.

```
IX=AND(IX*A, 34359738367)
RANUM(I)=IX/34359738368.0
```

CDC 6000 and CYBER

The peculiar feature of the CDC hardware is the lack of an integer multiply in the instruction set. An integer multiply is accomplished using the floating point multiply, which inherently restricts the integer product to 48 bits. Thus the maximum integer value we may obtain is $2^{48} - 1$. No overflow occurs in the sign field since only the lower 48 bits are returned as the result of an arithmetic operation [4]. This eliminates the necessity of checking for negative numbers.

Unsigned (Positive) Residue Method

No test is required to detect negative values. Division by 2^{48} produces the uniform variate.

```
IX=IX*IA
RANDUM(I)=IX/281474976710656.0
```

Logical Mask Method

The second method utilizes the logical operator OR to mask in the appropriate exponent $(1717)_8$. This produces an unnormalized floating point number with an exponent of 2^{-48}.

```
IX=IX*IA
RANUM(I)=OR(1717000000000000000B,IX)
```

COMPARISON OF GENERATION TIMES

An identical sequence of U(0,1) random variates is generated by each of these methods for the respective machines; the prominent issue is the relative "cost" of using one method over another. To obtain an objective comparison of each method, the following experimental conditions are imposed:

(1) Each method is inserted in a block of FORTRAN code, illustrated in Figure 1 without the accompanying output and documentation statements.

(2) Three different values are used for the linear multiplier (IA) on the SRU 1108 to remove any effect of a specific set of values. Each multiplier has been found to produce satisfactory results for the spectral test formulated by Knuth [5, p. 88]. Also, each multiplier has shown satisfactory performance when subjected to a battery of tests [7]. The initial value of the sequence (IX) on the SRU 1108 remains as 56329 for all runs. For the CDC 6000-CYBER series a single multiplier was used, and six independent samples (for both 5000 and 20000) were generated to obtain timing values.

(3) Sequences of length 5000 and 20,000 are generated based on the value assigned to MAXN.

(4) A timing subroutine (TIMER) on the SRU 1108 is inserted to record the execution time for the DO loop containing each generation method. While this time includes that necessary for the DO statement as well as the linkage necessary for TIMER, the time values provide accurate relative comparisons. An equivalent timing subroutine (SECOND) is used on the CDC 6000-CYBER.

(5) For the SRU 1107-1108 series, programs were executed on an SRU 1108, using FORTRAN V : LEVEL 6.0.

(6) For the CDC 6000-CYBER series, programs were executed on a CYBER-72, using FTN : VERSION 3.0.

The results are summarized in Table 1.

For the SRU 1107-1108 series, the logical mask method realizes the least execution time although its advantage over the masked sign method is rather small. The unsigned residue method requires approximately 32 percent more execution time than the logical mask method (83/63). Inclusion of the explicit test in the signed residue method imposes a cost of approximately 78 percent in execution time efficiency. These comparative values, based on the generation times for a sequence of length 5000, are increased slightly for a sequence of 20,000 values.

For the CDC 6000-CYBER series, the unsigned residue method requires approximately 30 percent more execution time than the logical mask method (135.6/105.5 and 546.8/419.0).

SRU 1107-1108

```
      .
      .
      .
      DIMENSION IS(2)
      DIMENSION RANUM(20000)
      IA=5**15
      IX=56329
      MAXN=5000
      CALL TIMER(IS)
      JB=5000*IS(1)+(4999-IS(2))
      DO 1 I=1,MAXN

      METHOD

1     RANUM(I)=IX/34359738368.0
      CALL TIMER(IS)
      JE=5000*IS(1)+(4999-IS(2))
      TIME=(JE-JB)*200
      .
      .
      .
```

CDC 6000 and CYBER

```
      .
      .
      .
      IA=553645B
      IX=1274321477413155B
      MAXN=5000
      T1=SECOND(A)
      DO 1 I=1,MAXN
      IX=IX*IA

      METHOD

1     CONTINUE
      T2=SECOND(A)
      TIME=T2-T1
      .
      .
      .
```

Figure 1. The Execution Time Comparison Programs (documentation and output statements omitted)

SRU 1107-1108 Timing Results

Multipliers (IA)	Sequence Length	Methods			
		Signed[3] Residue	Unsigned Residue	Masked Sign	Logical Mask
5**15	5000	111.8	83.4	64.4	63.4
	20000	457.2	335.4	259.4	247.8
3141592221	5000	112.8	83.4	64.8	61.0
	20000	450.2	334.6	259.4	247.8
2718281821	5000	112.8	83.2	64.6	60.8
	20000	448.6	334.6	259.6	244.8

CDC 6000-CYBER Timing Results

Multiplier (IA)	Sequence Length	Methods	
		Unsigned Residues	Logical Mask
553645B	5000	135.6	105.5
	20000	546.8	419.0

Table 1. Execution Times for Generators (in milliseconds)[4]

A COMPOSITE GENERATOR

In their paper cited above, Marsaglia and Bray note that composite generators can be constructed from simple one-line generators without inordinate loss of speed. The need for a composite generator, of course, is based on the desire to increase periodicity and/or to improve statistical performance. By combining two

[3]Marsaglia and Bray [1] give the speed of their generator for the SRU 1108 as 49,000/second. Our results for the signed residue method, indicated by L. R. Grosenbaugh to be the method used by him, give the speed as approximately 45,000/second. Accepting either value, the comparative results are not altered essentially.

[4]No array was maintained for values generated on the CYBER-72.

simple generators above, we obtain an efficient composite generator of the same type as proposed in the earlier work of MacLaren and Marsaglia [6]. The technique suggested by MacLaren and Marsaglia utilizes one generator to identify a location in a 128-element table. A second generator produces values to be inserted into, and later retrieved from, the location computed by the first generator. Of course, the table must be filled with values from the second generator before the combination procedure can begin.

The implementation of the composite generator requires four FORTRAN statements in addition to the division operation to return the uniform variate.

```
IX=AND(IX*IA, 34359738367)
IXJ=AND(IXJ*IAJ, 34359738367)
NUM=FLD(1,6,IXJ)+1
RANUM(I)=ITABLE(NUM)/34359738368.0
ITABLE(NUM)=IX
```

Note that the logical mask method is used to generate the values inserted into the table (IX) and to provide a value from which the table location (IXJ) is determined. The FLD function selects the six must significant bits to give a value in the interval [0,127].

Comparative execution times for our composite generator and that of Marsaglia and Bray [1] are shown in Table 2. For our generator the execution times are converted to a generation rate to facilitate comparison with the earlier results. Results are given for two runs at each level (5000 and 20000) since the timing values can vary slightly due to the manner of allocating overhead on the SRU 1108.

Method \ Number Generated	5000		20000	
	Run 1	Run 2	Run 1	Run 2
Nance & Overstreet (Execution time in ms)	123.8	115.2	485.8	465.4
Nance & Overstreet (Conversion of above values to generation rate)	40,000/ sec	43,500/ sec	41,000/ sec	43,000/ sec
MacLaren & Marsaglia (Composite In-Line)	16,500/sec			

Table 2. Comparison of Generation Times for Composite Generator

SUMMARY

Random number generators can be programmed in FORTRAN using several methods on 1's complement machines. We have presented several methods for the SRU 1107-1108 and CDC 6000 and CYBER series, the two most popular that utilize 1's complement arithmetic. We believe the logical mask method to be the most efficient technique based on execution time on both machines. This method requires only a little more than one-half the time of the signed residue method on the SRU 1107-1108 and a little more than two-thirds the time of the unsigned residue method on the CDC 6000-CYBER.

A fast composite generator in FORTRAN uses the logical mask method combined with the FLD function to implement the table approach of MacLaren and Marsaglia [6]. This generator proves to be more than twice as fast as a previous one for the SRU 1108, [1].

REFERENCES

1. Marsaglia, George and T. A. Bray, "One-Line Random Number Generators and Their Use in Combinations", Comm. ACM, 11(11): November 1968, 757-759.

2. Grosenbaugh, L. R., "More on Fortran Random Number Generators", Comm. ACM, 12(11): November 1969, 639.

3. FORTRAN V Programmers Reference Manual, Univac Data Processing Division UP-4060, 1966.

4. Control Data 6400/6600 Computer Systems Reference Manual, Pub. No. 60100000, Control Data Corporation, 1966.

5. Knuth, D. E., The Art of Computer Programming: Volume 2/Seminumerical Algorithms, Addison-Wesley, 1969.

6. MacLaren, M. D. and George Marsaglia, "Uniform Random Number Generators", J. ACM, 12(1): January 1965, 83-89.

7. Overstreet, Claude, Jr., "A FORTRAN V Package for Testing and Analysis of Pseudorandom Number Generators", Technical Report CP-72009, Computer Science/ Operations Research Center, Southern Methodist University, March 1972.

SUMMARY

Random number generators can be programmed in FORTRAN using several methods on 1's complement machines. We have presented several methods for the SRU 1107-1108 and CDC 6000 and CYBER series, the two most popular that utilize 1's complement arithmetic. We believe the logical mask method to be the most efficient technique based on execution time on both machines. This method requires only a little more than one-half the time of the signed residue method on the SRU 1107-1108 and a little more than two-thirds the time of the unsigned residue method on the CDC 6000-CYBER.

A fast composite generator in FORTRAN uses the logical mask method combined with the FLD function to implement the table approach of MacLaren and Marsaglia [6]. This generator proves to be more than twice as fast as a previous one for the SRU 1108, [1].