Bell Laboratories Cover Sheet for Technical Memorandum

1453

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **Implementing and Testing New Versions of a Good 48-bit Pseudo-random Number Generator**

Date: **September 4, 1979**

Other Keywords: **Random Number Generators**
Pseudo-random Sequences

TM: **79-1353-5**

USNSA

Author(s)
C. S. Roberts

Location
HO 4E-602

Extension
2044

Charging Case: **39394**
Filing Case: **39394**

ABSTRACT

We describe the design, implementation, testing, and use of drand48 — a good pseudo-random number generator based upon the linear congruential algorithm and 48-bit integer arithmetic. The drand48 subroutine is callable from C-language programs, and versions now exist for both the PDP-11 and VAX-11 computers. Given the same initialization value, both the PDP-11 and VAX-11 versions produce the identical sequence of pseudo-random numbers. Versions of drand48 for other computers or other programming languages clearly could be implemented, and some intermediate results have been tabulated to aid in testing and debugging such newly coded subroutines. Timing results for drand48 on the PDP-11/45, the PDP-11/70 and the VAX-11/780 are also presented and compared.

Pages Text: 5	Other: 2	Total: 7
No. Figures: 2	No. Tables: 3	No. Refs.: 5

SEE REVERSE SIDE FOR DISTRI...

DISTRIBUTION
(REFER GEI 13.9-3)

MEMORANDUM TO

WRENCE
EVEN M
N L
HARRY J,JR
,W P
KI,T P
JAMES A
N,G K
SKI,THOMAS G
BERKLEY A
BURY,S K
SON,K
,NEIL H E
TED,J TURNER
64 NAMES

ER SHEET ONLY TO

RRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ABATEMARCO,TERESA M
ABATE,JOSEPH
ACKERMAN,A FRANK
ACKROFF,JOHN M
ABO,ALFRED V
ALBAGLI,T E
ALCALAY,D
ALEXIS,A D,JR
ALLEN,ROBERT B
ALLISON,C E,JR
ALONSO,RAFAEL
AMITAY,M
ANDERSON,E E
ANDERSON,E E
ANDREWS,H J
ANTOLICK,DAVID E
APPELBAUM,MATTHEW A
ARCHER,RUSSELL E,JR
ARNOLD,JAMES O
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
ARVIDSON,W P
ASMUTH,RICHARD L
ASTHANA,ABHAYA
ATKINS,JOAN
AXELSON,A L
BABU,RAJESH RATILAL
BAKER,MITCHELL
BALDWIN,GEORGE L
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BALL,MARSHALL
BALTHUSE,MICHAEL A
BAROFSKY,ALLEN
BATTAGLIA,FRANCES

COVER SHEET ONLY TO

BAUER,ANDREW E
BAUER,BARBARA T
BAUER,d C
<BAUER,WOLFGANG F
<BADGE,C E
<BAXTER,LESLIE A
BECKER,JACOB I
BEIGHLEY,KEITH A
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENOWITZ,P
BENSING,JAMES EDWARD
BERENBAUM,ALAN
BERKEY,M A
BERNOSKE,BEVERLY G
BERNSTEIN,PAULA E
BERN,DAVID
BHATIA,RAJIV
BIANCHI,M E
BILASH,TIMOTHY D
BIREN,IRMA B
BISHOP,J DANIEL
BITTRICH,MARY E
BLAZIER,S D
BLINN,J C
BLOSSER,PATRICE A
BLUM,MARION
BOCK,NANCY E
BOECIE,JAMES E
BOEHM,KIM A
BOGART,P J
BOGART,THOMAS G
BOHANNI,L E
BOND,P C
BORG,KEVIN E
BORISON,ELLEN A
BOSE,DEBASISH
BOSTON,RONALD E
BOSWELL,PAULA S
BOURNE,STEPHEN E
BOWERMAN,REBECCA ELAINE
BRADFORD,EDWARD G
BRADLEY,M HELEN
BRAINARD,E C
BRANDT,RICHARD S
BRANTLEY,JOAN T
BRAUN,DAVID A
BREILAND,JOHN E
BRENSKI,EDWIN F
BRESLER,RENEE A
BRITT,DARREN D
BRONSTEIN,N
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BRUMAN,INNA
BROWN,ALLINGTON L
BROWN,BA S
BROWN,W T

COVER SHEET ONLY TO

BUCK,I D
BURGESS,JOHN T,JR
BURIC,MILORAD E
BURKE,MARY P
BURKE,E J
BURNETTE,W A
BURNETT,DAVID S
BURNET,ROSE M
BURGPP,STEVEN J
BUTLETT,DARRELL L
<BUTZIEN,PAUL F
BYORICK,ROBERT S
CALLAHAN,E L
CAMPBELL,JERRY E
CAMPBELL,MICHAEL E
CANADAY,RUDD E
CANDREA,RONALD D
CAREY,J E
CARPENTER,THOMAS L,III
CARRAN,JOHN E
CARR,DAVID C
CARTER,DONALD E
CHAFFEE,E F
CHAI,D T
CHANG,JO-MEI
CHAO,C
CHAPMAN,LESLEY A
CHAPPELL,S G
CHARLES,JOYCE
CHEE,T
CHELLIS,ALICIA L
CHENG-QUISPE,ENRIQUE
CHENG,Y
CHEN,ROBERT
CHEN,STEPHEN
CHERBY,LORINDA L
CHE,HER-DAH
CHODROW,M M
CHRISTENSON,DENNIS A
CHUNG,TSUNG-JEN BEN
CICHINSKI,STEVEN
CIEMINSKI,DEBRA F
CLARK,DAVID L
CLIFFORD,COURTENAY B
CLINE,LAUREL M I
CLINE,TERRY W
CLOUTIER,J
COHEN,AARON S
COHEN,DAVID
COHEN,HARVEY
COHOON,JAMES P
COLEMAN,ELAINE
<COLE,LOUIS F
COLE,MARILYN O
COLLICOTT,E B
CONNAGHAN,ROBERT
CONNERS,RONALD E
COOK,T J
COOPER,MICHAEL H
COPP,DAVID H
COTTRELL,JENNIE L
<COURTNEY-PRATT,J S

ITHOUT PREFIX
GIVEN BELOW)

813

AUTHOR > CITED AS REFERENCE < REQUESTED BY READE
TED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIA

CIFICATION.....

MC TO:
13-EIR 11-EXD 15-EXE 16-EXD 1353-TA 13

128-SUP

ET TO:
TS 127-SUP

= MATHEMATICAL SUBROUTINES

ROBERTS,C S
HO 4E602

PLEASE SEND A COMPLETE COPY
OTHER SIDE TM-79-
NC ENVELOPE WILL BE NEEDED IF TOTAL PA
SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE SHOW
REQUEST TO THE CORRESPONDENCE FI

PLE

D

GET A COMPLETE COPY:

BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
CIRCLE THE ADDRESS AT RIGHT. USE NC ENVELOPE.

Bell Laboratories

subject: **Implementing and Testing New Versions of a Good 48-bit Pseudo-random Number Generator**
Case: **39394**
File: **39394**

date: **September 4, 1979**

from: **C. S. Roberts**
HO 1353
4E-602 x2044

TM: **79-1353-5**

MEMORANDUM FOR FILE

1. INTRODUCTION

The work described in this paper arose when one day the author found himself in need of a good pseudo-random number generator which would execute and produce identical results on two different computers (the 16-bit PDP-11 and the 32-bit VAX-11 both manufactured by Digital Equipment Corporation). Good pseudo-random number generators often require multiple precision arithmetic, and speed of execution can be a critical factor; hence, their implementation usually is done in assembly language and is dependent upon the word length of the computer. If this is not enough of a barrier to portability, add the fact that the recoding of a pseudo-random number generator for a different computer is an error-prone endeavor. The author has himself found bugs in several pseudo-random number subroutines which were supposedly "correctly" coded. (Being facetious, do such bugs make the output from a generator more or less random?)

This paper does not present a magical method to eliminate these difficulties. However, it does present enough data and intermediate results so that a person may code on any computer a very good generator based upon 48-bit integer arithmetic and then verify whether or not it is operating correctly. In addition, we describe proper usage of routines *drand48*, *lrand48* and *mrand48* — C-language callable functions to generate pseudo-random numbers — available at Bell Laboratories on both the PDP-11 and VAX-11 computers.

The pseudo-random number generator considered in this paper is based upon the well known linear congruential algorithm, e.g., see Section 3.2.1 of Knuth [1]. The next number in the pseudo-random sequence is generated according to the formula

$$X_{n-1} = (aX_n + c) \bmod m \qquad n \geqslant 0.$$

We choose the value of m to be 2^{48}, hence 48-bit integer arithmetic is required. The values of the multiplier a and the addend c are chosen as follows:

$$a = 5\text{DEECE66D}(\textit{hexadecimal}) = 273673163155(\textit{octal})$$
$$c = \text{B}(\textit{hexadecimal}) = 13(\textit{octal}).$$

The above value of a was one of the multiplier values judged by Coveyou and MacPherson [2] to be satisfactory according to the "spectral test". The above values of a, c and m ensure that the generated pseudo-random sequence will have the maximum possible period, i.e., 2^{48} (see Theorem A, Section 3.2.1.2 of Knuth [1]).

2. PSEUDO-RANDOM GENERATOR RESULTS

The pseudo-random sequence generator described and specified in Section 1 was coded in assembly language three times — once on the PDP-11 using the 16-bit integer arithmetic instructions *mul* and *add*[3], once on the PDP-11 using the 64-bit floating-point arithmetic instructions *mulf* and *addf*[3], and once on the VAX-11/780 using the 32-bit integer extended multiply and add instruction *emul*[4]. After a bit of debugging, all three independent versions finally produced the identical sequence of 48-bit pseudo-random integers. For purposes of comparison with future newly coded subroutines, a portion of that sequence is presented below (in hexadecimal) as Table 1.

TABLE 1. Portion of the pseudo random sequence X_i of 48-bit integers

1234ABCD330E	657EB7255101	D72A0C966378	5A743C062A23	72534ABF62F2
5195D97A8D15	E2ECF94AEFFC	03FD3CD49657	9586EFCA2D16	28CC61DEF669
623B341D40C0	B0E5A9A111CB	0F1160B4F57A	E65CDA1020FD	29DE25BD59C4
28B8E8F5507F	8876EDD9601E	9AA93190E0D1	952BC3577F08	451CD3C24673
63F661075102	4B1C4CBD49E5	BE0C7218348C	4C6C2C9427A7	135676A8EC26
67ACF11EB039	DB7D1EF03E50	F124D606681B	A9AF4526958A	D8B2A2FFA7CD
00B48E98A054	765E7C77BBCF	8858368AF12E	C9B2484004A1	43FF29D69E98
FB95A6FE16C3	4E897866E312	99D1A468DAB5	9BD4C9FFBD1C	3662639AACF7

Since it is common practice to treat the output from a pseudo-random number generator as a pure binary fraction, thus yielding a uniform distribution over the interval 0 to 1, the numbers in Table 1 were so treated and then multiplied by 4096 to yield the decimal integers in Table 2. More precisely, each integer Y_i in Table 2 was computed from the corresponding value of X_i in Table 1 by the formula

$$Y_i = \lfloor (X_i)(2^{-48})(4096) \rfloor = \lfloor 2^{-36} X_i \rfloor .$$

For some purposes, Table 2 may be more convenient or appropriate than Table 1.

TABLE 2. Pseudo-random integers Y_i corresponding to the X_i in Table 1

291	1623	3442	1447	1829	1305	3630	63	2392	652
1571	2830	241	3685	669	651	2183	2474	2386	1105
1599	1201	3040	1222	309	1658	3511	3858	2714	3467
11	1893	2181	3227	1087	4025	1256	2461	2493	870
3628	1247	622	1383	1587	2636	3086	2472	2177	1881
2672	1340	3876	1507	3866	30	2115	1117	99	2424
839	3595	243	1068	1240	3651	2040	2908	1173	3542
2767	1877	3930	3173	1542	936	1452	1230	2743	2944

Tables 1 and 2 should be valuable to a person attempting to code and test a new version of the generator subroutine. After initializing with X_0=1234ABCD330E (hexadecimal), the new version, if correct, should reproduce Table 1 (or Table 2). The fact that three independent codings of the 48-bit linear congruential algorithm gave the same results gives the author substantial confidence that Tables 1 and 2 are indeed correct.

The author has found that one quick and effective way of evaluating the overall quality of any pseudo-random random number generator is to display the output bits on a bilevel display device, and this has been done in Figure 1 for the generator defined in Section 1 of this paper. Figure 1 was generated using only the leftmost 32 bit of each 48-bit X_i; 2^{13} values of X_i were generated thus giving a total of 2^{18} bits. These bits were displayed on a 512 × 512 CRT display device with a *1* being displayed as white and a *0* being displayed as black. Good quality pseudo-random number generators produce displays similar in appearance to Figure 1 — random salt and pepper effect with no visible patterning. For comparison, Figure 2 shows the display produced by an inferior generator. Note the prominent nonrandom patterns visible in Figure 2,

which was produced using the leftmost eight bits of output from the library routine *rand* described in Section 3, Subroutines, of the UNIX* Programmer's Manual.

3. SUBROUTINES FOR USE WITH THE C PROGRAMMING LANGUAGE

As an example of how the calling sequences to the pseudo-random number generator might be designed, we now describe some routines which were coded for use with the C programming language [5]. So far versions of these routines have been implemented only for the PDP-11 and VAX-11 computers. However, versions for other computers which support the C language clearly could be created if the need warranted.

The pseudo-random number generator was implemented in assembly language as one subroutine having six entry points. Three of the entry points are for subroutine initialization; three are for generating the next pseudo-random X_i in the sequence. A program must first call one of the initialization entry points — either srand48, seed48 or lcong48; subsequently any of the entry points drand48, lrand48 or mrand48 may be called multiple times to retrieve successive X_i converted to an appropriate data format. Some of the calling sequences assume a computer whose word size is an integer multiple of 8 bits.

3.1 Function drand48

Example C usage:

```
double fnext, drand48();
fnext = drand48();
```

The next pseudo-random number is returned as a positive valued binary fraction in double precision floating point format; i.e., the value returned is $2^{-48}X_i$. The values returned are uniformly distributed over the interval [0,1). Thus, in the above C-code example, the value range for fnext is $0 \leqslant$ fnext < 1.

3.2 Function lrand48

Example C usage:

```
long int lnext, lrand48();
lnext = lrand48();
```

The next pseudo-random number is returned as a positive valued integer in long integer format. The long integer is formed by just taking the leftmost 31 bits of X_i, i.e., the value returned is $\lfloor 2^{-17}X_i \rfloor$. In the above C-code example, the value range for lnext is $0 \leqslant$ lnext $< 2^{31}$.

3.3 Function mrand48

Example C usage:

```
long int mnext, mrand48();
mnext = mrand48();
```

The next pseudo-random number is returned as a signed integer in long integer format. The long integer is formed by just taking the leftmost 32 bits of X_i to be a signed integer represented in two's-complement format. Hence, the leftmost bit of X_i determines the sign of the output value. In the above C-code example, the value range for mnext is $-2^{31} \leqslant$ mnext $< 2^{31}$.

---

* UNIX is a trademark of Bell Laboratories.

3.4 Function srand48

Example C usage:

```
long int seedval;
seedval = 0x1234ABCD;
srand48(seedval);
```

This is an initialization entry point which sets the value of X_0, the multiplier a and the addend c are set to the values specified in Section 1. The leftmost 32 bits of X_0 are taken from the argument passed to srand48 when it is called (seedval in the above C-code example). The rightmost 16 bits of X_0 are arbitrarily set to 0x330E. Hence, the above C-code example sets the value of X_0 to 0x1234ABCD330E.

3.5 Function seed48

Example C usage:

```
short int seed16v[3];
seed16v[0] = 0x330E;
seed16v[1] = 0xABCD;
seed16v[2] = 0x1234;
seed48(seed16v);
```

This is an initialization entry point which sets the value of X_0 to the 48 bits specified by the argument passed to seed48; the multiplier a and the addend c are set to the values specified in Section 1. The argument is an array of three 16-bit integers. The above C-code example sets the value of X_0 to 0x1234ABCD330E.

3.6 Function lcong48

Example C usage:

```
short int param[7];
param[0]=0x330E; param[1]=0xABCD; param[2]=0x1234;
param[3]=0xE66D; param[4]=0xDEEC; param[5]=0x5;
param[6]=0xB;
lcong48(param);
```

This is an initialization entry point which sets the values of X_0, a and c; hence different 48-bit linear congruential generators may be created by specifying different values for the multiplier a and the addend c. The argument passed to lcong48 is an array of seven 16-bit integers. The first three specify a 48-bit value of X_0; the next three specify a 48-bit value of the multiplier a, and the last one specifies a 16-bit value of the addend c. Hence, the above C-code example sets X_0=0x1234ABCD330E, a=0x5DEECE66D and c=0xB.

4. TIMING RESULTS

For people who are interested in such things, Table 3 presents the time required to generate, using function drand48, 10^6 pseudo-random numbers on four different computer hardware configurations.

TABLE 3. Time required to generate 10^6 pseudo-random numbers

PDP-11/45	440 secs
PDP-11/45 with Fabritek cache	340
PDP-11/70	162
VAX-11/780	100

More precisely, Table 3 gives the time required to execute the following short C-language program:

```
main( ) {
register int i, j, h;
double nfd, drand48( );
int lli, llj;
short int nn[500];
long int seedval;
seedval = 0x1234ABCD; srand48(seedval);
for(i=0; i<500; i++) nn[i]=0;
lli = 1000; llj = 1000;
nfd = 500;
for(i=0; i<lli; i++) for(j=0; j<llj; j++) {
 h = nfd * drand48( );
 nn[h]++;
 }
}
```

Execution of drand48 accounts for 80%, approximately, of the times listed in Table 3. For the timing tests on the PDP-11/45 and PDP-11/70, our version of drand48 that employs the floating-point arithmetic instructions was used since it is substantially faster than the other version which employs the integer arithmetic instructions. The first three entries in Table 3 represent separate executions of the same binary machine code; hence, the time differences reflect brute-force speed differences of the processor and/or memory hardware. The comparison with the VAX-11/780 is, however, more subtle since a complete recoding of drand48 is involved here. Essentially, when comparing the VAX-11 with the PDP-11 in Table 3, we are comparing the time required to accomplish the identical *function* on both computers using a near optimally coded version of drand48 on each machine. The faster time for the VAX-11/780 must be attributed to at least two factors: a more powerful and capable instruction set and faster basic hardware modules.

C. S. Roberts

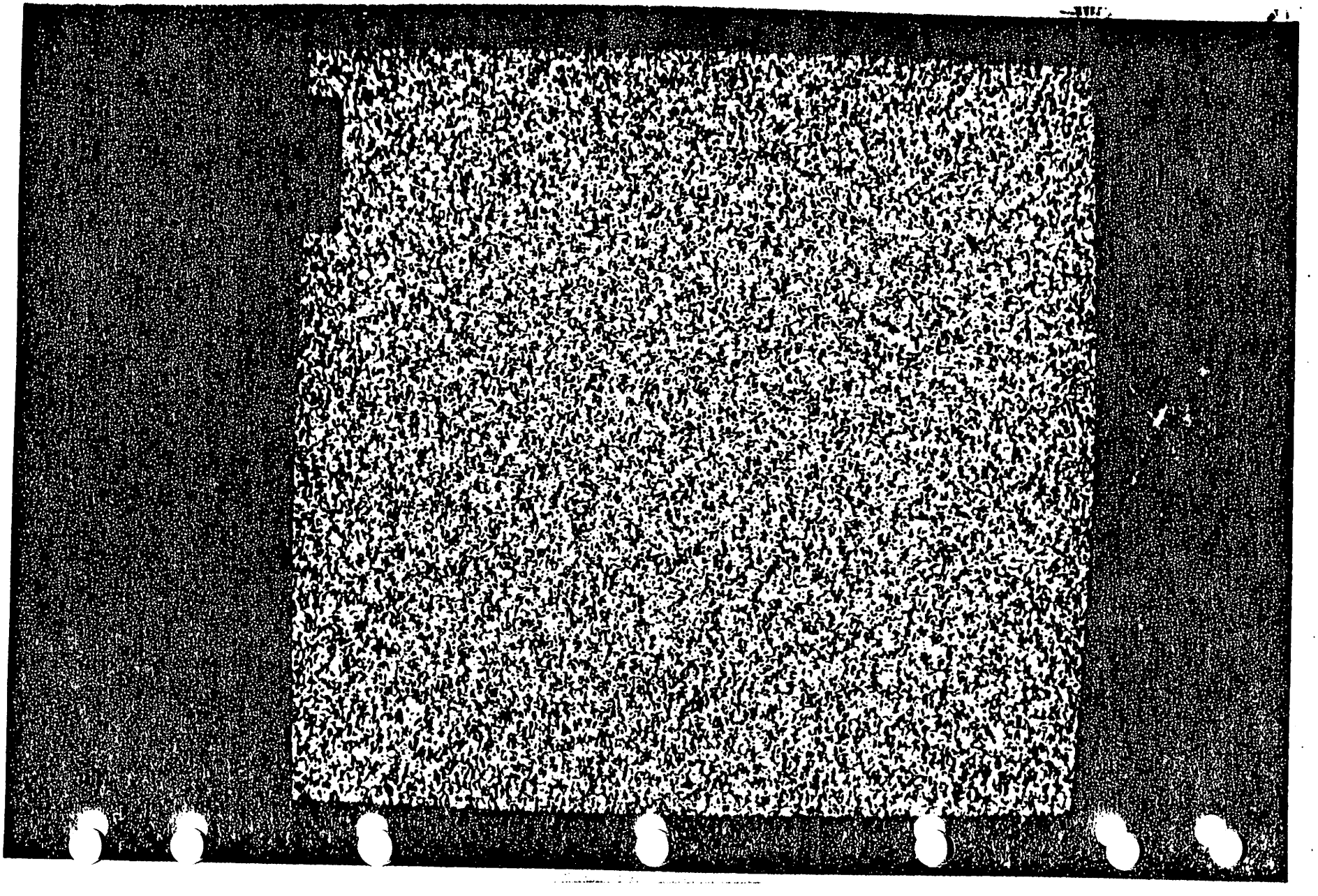
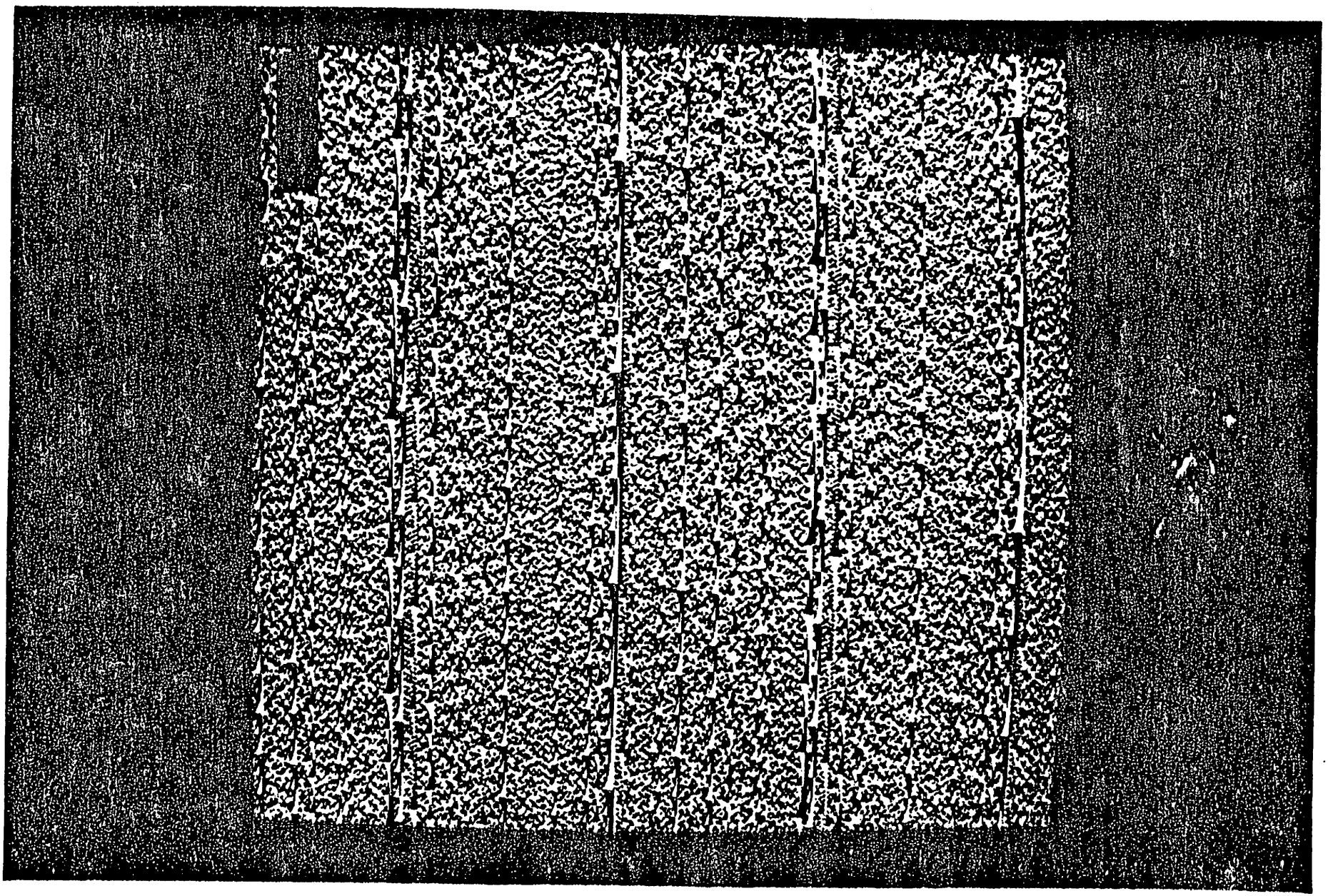
HO-1353-CSR-jk

Atts.
References 1 through 5
Photographs:
B79-3334HO (Figure 1)
B79-3335HO (Figure 2)

REFERENCES

1. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass., 1969.
2. R. R. Coveyou and R. D. MacPherson, 'Fourier Analysis of Uniform Random Number Generators', *J. Assoc. Comp. Mach.*, **14**, 1, 100-119(1967).
3. *PDP-11/70 Processor Handbook*, Digital Equipment Corporation, Maynard, Mass., 1976.
4. *VAX-11/780 Architecture Handbook Vol. 1*, Digital Equipment Corporation, Maynard, Mass., 1977.
5. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

Bell Laboratories CASE NO 39394 FIGURE 1 NEG NO B79-3334H0



Bell Laboratories CASE NO 39394 FIGURE 2 NEG NO B79-3335H0