# Random-Number Generation


- Desired properties of a good generator
- Linear-congruential generators
- Tausworthe generators
- Survey of random number generators
- Seed selection
- Myths about random number generation

# Random-Number Generation

- Random Number = Uniform (0, 1)
- Random Variate = Other distributions
  = Function(Random number)

# A Sample Generator

$$x_n = f(x_{n-1}, x_{n-2}, \ldots)$$

- For example,

  $$x_n = 5x_{n-1} + 1 \mod 16$$

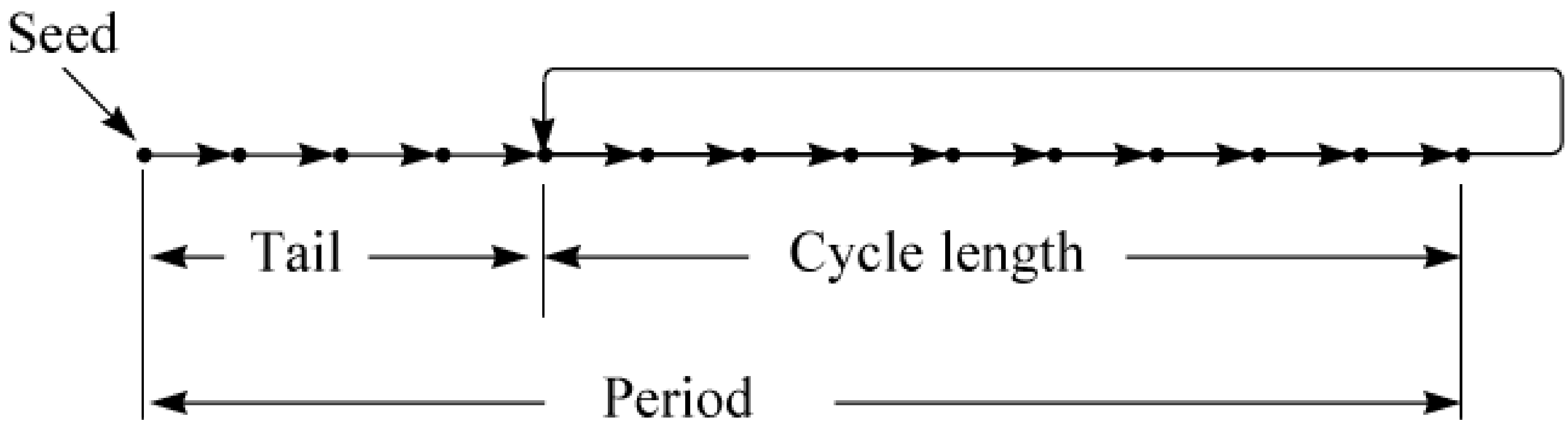
- Starting with $x_0$=5:

  $$x_1 = 5(5) + 1 \mod 16 = 26 \mod 16 = 10$$

- The first 32 numbers obtained by the above procedure 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.
- By dividing x's by 16:
  0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125, 0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375, 0.2500, 0.3125.

# Terminology

- **Seed** = $x_0$
- **Pseudo-Random**: Deterministic yet would pass randomness tests
- Fully Random: Not repeatable
- **Cycle length**, **Tail**, **Period**



# Desired Properties of a Good Generator

- It should be efficiently computable.
- The period should be large.
- The successive values should be independent and uniformly distributed

# Types of Random-number Generators

- Linear congruential generators
- Tausworthe generators
- Extended Fibonacci generators
- Combined generators

# Linear-Congruential Generators

- Discovered by D. H. Lehmer in 1951
- The residues of successive powers of a number have good randomness properties.

$$x_n = a^n \bmod m$$

Equivalently,

$$x_n = a x_{n-1} \bmod m$$

*a* = multiplier

*m* = modulus

# Linear-Congruential Generators (Cont)

- Lehmer's choices: $a = 23$ and $m = 10^8+1$
- Good for ENIAC, an 8-digit decimal machine.
- Generalization:

$$x_n = ax_{n-1} + b \bmod m$$

- Can be analyzed easily using the theory of congruences
  $\Rightarrow$ Mixed Linear-Congruential Generators
  or Linear-Congruential Generators (LCG)
- Mixed = both multiplication by *a* and addition of *b*

# Selection of LCG Parameters

- $a$, $b$, and $m$ affect the period and autocorrelation
- The modulus $m$ should be large.
- The period can never be more than $m$.
- For mod $m$ computation to be efficient, $m$ should be a power of 2 $\Rightarrow$ Mod $m$ can be obtained by truncation.

# Selection of LCG Parameters (Cont)

- If *b* is nonzero, the maximum possible period *m* is obtained if and only if:
  - Integers *m* and *b* are relatively prime, that is, have no common factors other than 1.
  - Every prime number that is a factor of *m* is also a factor of *a*-1.
  - If integer *m* is a multiple of 4, *a*-1 should be a multiple of 4.
  - Notice that all of these conditions are met if $m=2^k$, a = 4c + 1, and *b* is odd. Here, *c*, *b*, and *k* are positive integers.

# Period vs. Autocorrelation

- A generator that has the maximum possible period is called a full-period generator.

$$x_n = (2^{34}+1)x_{n-1}+1 \mod 2^{35}$$

$$x_n = (2^{18}+1)x_{n-1}+1 \mod 2^{35}$$

- Lower autocorrelations between successive numbers are preferable.
- Both generators have the same full period, but the first one has a correlation of 0.25 between $x_{n-1}$ and $x_n$, whereas the second one has a negligible correlation of less than $2^{-18}$

# Multiplicative LCG

- Multiplicative LCG: $b$=0

$$x_n = ax_{n-1} \bmod m$$

- Two types:

  $m = 2^k$

  $m \neq 2^k$

# Multiplicative LCG with $m=2^k$

- $m = 2^k \Rightarrow$ trivial division
  $\Rightarrow$ Maximum possible period $2^{k-2}$
- Period achieved if multiplier a is of the form $8i$§ 3, and the initial seed is an odd integer
- One-fourth the maximum possible may not be too small
- Low order bits of random numbers obtained using multiplicative LCG's with $m=2^k$ have a cyclic pattern

# Example 26.1a

$$x_n = 5x_{n-1} \bmod 2^5$$

- Using a seed of $x_0$=1:

  5, 25, 29, 17, 21, 9, 13, 1, 5,…

  Period = 8 = 32/4
- With $x_0$ = 2, the sequence is: 10, 18, 26, 2, 10,…

  Here, the period is only 4.

# Example 26.1b

- Multiplier not of the form $8i \pm 3$:

$$x_n = 7x_{n-1} \bmod 2^5$$

- Using a seed of $x_0 = 1$, we get the sequence:
  7, 17, 23, 1, 7,…
- The period is only 4

# Multiplicative LCG with $m \neq 2^k$

- Modulus $m$ = prime number

  With a proper multiplier $a$, period = $m$-1

  Maximum possible period = $m$
- If and only if the multiplier a is a *primitive root* of the modulus $m$
- $a$ is a primitive root of $m$ if and only if $a^n \bmod m \neq 1$ for $n$ = 1, 2, …, $m$-2.

# Example 26.2

$$x_n = 3x_{n-1} \bmod 31$$

- Starting with a seed of $x_0$=1:

  1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, …

  The period is 30

  ⇒ 3 is a primitive root of 31

- With a multiplier of *a* = 5: 1, 5, 25, 1,…

  The period is only 3 ⇒ 5 is not a primitive root of 31

  $$5^3 \bmod 31 = 125 \bmod 31 = 1$$

- Primitive roots of 31= 3, 11, 12, 13, 17, 21, 22, and 24.

# Schrage's Method

- PRN computation assumes:
  - No round-off errors, integer arithmetic and no overflows $\Rightarrow$ **C**an't do it in BASIC
  - Product a $x_{n-1}$ > Largest integer $\Rightarrow$ Overflow
- Identity: $ax \bmod m = g(x) + mh(x)$

Where: $g(x) = a(x \bmod q) - r(x \text{ div } q)$

And: $h(x) = (x \text{ div } q) - (ax \text{ div } m)$

Here, q = m div a, r = m mod a

`A div B' = dividing A by B and truncating the result.

- For all x's in the range 1, 2, …, m-1, computing g(x) involves numbers less than m-1.
- If r < q, h(x) is either 0 or 1, and it can be inferred from g(x);

h(x) is 1 if and only if g(x) is negative.

# Example 26.3

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

- $2^{31}$-1 = 2147483647 = prime number
- $7^5$ = 16807 is one of its 534,600,000 primitive roots
- The product a $x_{n-1}$ can be as large as 16807× 2147483647 ≈ 1.03× $2^{45}$.
- Need 46-bit integers

$$\begin{aligned} a &= 16807 \\ m &= 2147483647 \\ q &= m \text{ div } a = 2147483647 \text{ div } 16807 = 12773 \\ r &= m \bmod a = 2147483647 \bmod 16807 = 2836 \end{aligned}$$

- For a correct implementation, $x_0 = 1 \Rightarrow x_{10000} = 1{,}043{,}618{,}065$.

# Generator Using Integer Arithmetic

```
FUNCTION Random(VAR x:INTEGER) : REAL;

CONST
  a = 16807;         (* Multiplier *)
  m = 2147483647;    (* Modulus *)
  q = 127773;        (* m div a *)
  r = 2836;          (* m mod a *)

VAR
  x_div_q, x_mod_q, x_new: INTEGER;

BEGIN
  x_div_q := x DIV q;
  x_mod_q := x MOD q;
  x_new := a*x_mod_q - r*x_div_q;
  IF x_new > 0 THEN x := x_new ELSE x := x_new + m;
  Random := x/m;
END;
```

# Generator Using Real Arithmetic

```
FUNCTION Random(VAR x:DOUBLE) : DOUBLE;

CONST
  a = 16807.0D0;        (* Multiplier *)
  m =  2147483647.0D0;  (* Modulus *)
  q = 127773.0D0;       (* m div a *)
  r = 2836.0D0;         (* m mod a *)

VAR
  x_div_q, x_mod_q, x_new: DOUBLE;

BEGIN
  x_div_q := TRUNC(x/q);
  x_mod_q := x-q*x_div_q;
  x_new := a*x_mod_q - r*x_div_q;
  IF x_new > 0.0D0 THEN x := x_new ELSE x := x_new + m;
  Random := x/m;
END;
```

# Tausworthe Generators

- Need long random numbers for cryptographic applications
- Generate random sequence of binary digits (0 or 1)
- Divide the sequence into strings of desired length
- Proposed by Tausworthe (1965)

$$b_n = c_{q-1}b_{n-1} \oplus c_{q-2}b_{n-2} \oplus c_{q-3}b_{n-3} \oplus \cdots \oplus c_0 b_{n-q}$$

Where $c_i$ and $b_i$ are binary variables with values of 0 or 1, and $\oplus$ is the exclusive-or (mod 2 addition) operation.

- Uses the last *q* bits of the sequence
  $\Rightarrow$ autoregressive sequence of order *q* or AR(*q*).
- An AR(*q*) generator can have a maximum period of $2^q$-1.

# Tausworthe Generators (Cont)

- $D$ = delay operator such that $Db(n) = b(n+1)$

$$D^q b(i-q) = c_{q-1}D^{q-1}b(i-q) + c_{q-2}D^{q-2}b(i-q) + \cdots + c_0 b(i-q) \bmod 2$$

$$D^q - c_{q-1}D^{q-1} - c_{q-2}D^{q-2} - \cdots - c_0 = 0 \bmod 2$$
$$D^q + c_{q-1}D^{q-1} + c_{q-2}D^{q-2} + \cdots + c_0 = 0 \bmod 2$$

- **Characteristic polynomial:**

  $$x^q + c_{q-1}x^{q-1} + c_{q-2}x^{q-2} + \cdots + c_0$$

- The period is the smallest positive integer $n$ for which $x^n$-1 is divisible by the characteristic polynomial.
- The maximum possible period with a polynomial of order $q$ is $2^q$-1. The polynomials that give this period are called **primitive polynomials**.

# Example 26.4

$x^7+x^3+1$

- Using $D$ operator in place of $x$:

$$D^7b(n) + D^3b(n) + b(n) = 0 \bmod 2$$

Or:

$$b_{n+7} + b_{n+3} + b_n = 0 \bmod 2 \quad n = 0, 1, 2, \ldots$$

- Using the exclusive-or operator

$$b_{n+7} \oplus b_{n+3} \oplus b_n = 0 \quad n = 0, 1, 2, \ldots$$

Or:

$$b_{n+7} = b_{n+3} \oplus b_n \quad n = 0, 1, 2, \ldots$$

- Substituting $n$-7 for $n$:

$$b_n = b_{n-4} \oplus b_{n-7} \quad n = 7, 8, 9, \ldots$$

# Example 26.4 (Cont)

- Starting with $b_0 = b_1 = \cdots = b_6 = 1$:

$$\begin{aligned} b_7 &= b_3 \oplus b_0 = 1 \oplus 1 = 0 \\ b_8 &= b_4 \oplus b_1 = 1 \oplus 1 = 0 \\ b_9 &= b_5 \oplus b_2 = 1 \oplus 1 = 0 \\ b_{10} &= b_6 \oplus b_3 = 1 \oplus 1 = 0 \\ b_{11} &= b_7 \oplus b_4 = 0 \oplus 1 = 1 \end{aligned}$$

- The complete sequence is:

  1111111 0000111 0111100 1011001 0010000 0010001 0011000 1011101 0110110 0000110 0110101 0011100 1111011 0100001 0101011 1110100 1010001 1011100 0111111 1000011 1000000.

- Period = 127 or $2^7$-1 bits

⇒ The polynomial $x^7+x^3+1$ is a primitive polynomial.

# Combined Generators

1. Adding random numbers obtained by two or more generators.

$w_n = (x_n + y_n) \bmod m$

For example, L'Ecuyer (1986):

$$x_n = 40014 x_{n-1} \bmod 2147483563$$

$$y_n = 40692 y_{n-1} \bmod 2147483399$$

This would produce:

$$w_n = (x_n - y_n) \bmod 2147483562$$

Period = $2.3 \times 10^{18}$

# Combined Generators (Cont)

Another Example: For 16-bit computers:

$$w_n = 157 w_{n-1} \bmod 32363$$

$$x_n = 146 x_{n-1} \bmod 31727$$

$$y_n = 142 y_{n-1} \bmod 31657$$

Use:

$$v_n = (w_n - x_n + y_n) \bmod 32362$$

This generator has a period of $8.1 \times 10^{12}$.

# Combined Generators (Cont)

2. Exclusive-or random numbers obtained by two or more generators.

3. Shuffle. Use one sequence as an index to decide which of several numbers generated by the second sequence should be returned.

# Combined Generators (Cont)

- Algorithm M:

a) Fill an array of size, say, 100.

b) Generate a new $y_n$ (between 0 and $m$-1)

c) Index $i=1+100\ y_n/m$

d) $i$th element of the array is returned as the next random number

e) A new value of $x_n$ is generated and stored in the $i$th location

# Survey of Random-Number Generators

- A currently popular multiplicative LCG is:

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

  - Used in:
    - SIMPL/I system (IBM 1972),
    - APL system from IBM (Katzan 1971),
    - PRIMOS operating system from Prime Computer (1984), and
    - Scientific library from IMSL (1980)
  - $2^{31}$-1 is a prime number and $7^5$ is a primitive root of it $\Rightarrow$ Full period of $2^{31}$-2.
- This generator has been extensively analyzed and shown to be good.
- Its low-order bits are uniformly distributed.

# Survey of RNG's (Cont)

- Fishman and Moore (1986)'s exhaustive search of $m=2^{31}-1$:

$$x_n = 48271 x_{n-1} \bmod (2^{31}-1)$$

$$x_n = 69621 x_{n-1} \bmod (2^{31}-1)$$

- SIMSCRIPT II.5 and in DEC-20 FORTRAN:

$$x_n = 630360016 x_{n-1} \bmod (2^{31}-1)$$

# Survey of RNG's (Cont)

- ``RANDU" (IBM 1968): Very popular in the 1960s:

$$x_n = (2^{16} + 3)x_{n-1} \bmod 2^{31}$$

  - Modulus and the multiplier were selected primarily to facilitate easy computation.
  - Multiplication by $2^{16}$+3=65539 can be easily accomplished by a few shift and add instructions.
  - Does not have a full period and has been shown to be flawed in many respects.
  - Does not have good randomness properties (Knuth, p 173).
  - Triplets lie on a total of 15 planes
    $\Rightarrow$ Unsatisfactory three-distributivity
  - Like all LCGs with m=$2^k$, the lower order bits of this generator have a small period. RANDU is no longer used

# Survey of RNG's (Cont)

- Analog of RANDU for 16-bit microprocessors:

$$x_n = (2^8 + 3)x_{n-1} \bmod (2^{15})$$

  - This generator shares all known problems of RANDU
  - Period = only a few thousand numbers
    ⇒ not suitable for any serious simulation study
- University of Sheffield Pascal system for Prime Computers:

$$x_n = 16807x_{n-1} \bmod 2^{31}$$

  - 16807 ≠ 8i§ 3 ⇒ Does not have the maximum possible period of $2^{31}$-2.
  - Used with a shuffle technique in the subroutine UNIFORM of the SAS statistical package

# Survey of RNG's (cont)

- SIMULA on UNIVAC uses the following generator:

  $$x_n = 5^{13} x_{n-1} \bmod 2^{35}$$

  - Has maximum possible period of $2^{33}$, Park and Miller (1988) claim that it does not have good randomness properties.
- The UNIX operating system:

  $$x_n = (1103515245 x_{n-1} + 12345) \bmod 2^{32}$$

  - Like all LCGs with $m=2^k$, the binary representation of $x_n$'s has a cyclic bit pattern

# Seed Selection

- **Multi-stream simulations**: Need more than one random stream
  - Single queue ⇒ Two streams
    = Random arrival and random service times

1. Do not use zero. Fine for mixed LCGs.
   But multiplicative LCG or a Tausworthe LCG will stick at zero.
2. Avoid even values. For multiplicative LCG with modulus $m=2^k$, the seed should be odd. Better to avoid generators that have too many conditions on seed values or whose performance (period and randomness) depends upon the seed value.
3. Do not subdivide one stream.

# Seed Selection (Cont)

4. Do not generate successive seeds: $u_1$ to generate inter-arrival times, $u_2$ to generate service time ⇒ Strong correlation
5. Use non-overlapping streams.
   Overlap ⇒ Correlation, e.g., Same seed ⇒ same stream
6. Reuse seeds in successive replications.
7. Do not use random seeds: Such as the time of day.
   Can't reproduce. Can't guaranteed non-overlap.
8. Select $\{u_0, u_{100,000}, u_{200,000}, \ldots\}$

$$x_n = a^n x_0 + \frac{c(a^n - 1)}{a - 1} \bmod m$$

# Table of Seeds

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

| $x_{100000i}$ | $x_{100000(i+1)}$ | $x_{100000(i+2)}$ | $x_{100000(i+3)}$ |
|---:|---:|---:|---:|
| 1 | 46,831,694 | 1,841,581,359 | 1,193,163,244 |
| 727,633,698 | 933,588,178 | 804,159,733 | 1,671,059,989 |
| 1,061,288,424 | 1,961,692,154 | 1,227,283,347 | 1,171,034,773 |
| 276,090,261 | 1,066,728,069 | 209,208,115 | 554,590,007 |
| 721,958,466 | 1,371,272,478 | 675,466,456 | 1,095,462,486 |
| 1,808,217,256 | 2,095,021,727 | 1,769,349,045 | 904,914,315 |
| 373,135,028 | 717,419,739 | 881,155,353 | 1,489,529,863 |
| 1,521,138,112 | 298,370,230 | 1,140,279,430 | 1,335,826,707 |
| 706,178,559 | 110,356,601 | 884,434,366 | 962,338,209 |
| 1,341,315,363 | 709,314,158 | 591,449,447 | 431,918,286 |
| 851,767,375 | 606,179,079 | 1,500,869,201 | 1,434,868,289 |
| 263,032,577 | 753,643,799 | 202,794,285 | 715,851,524 |

# Myths About Random-Number Generation

1. *A complex set of operations leads to random results*. It is better to use simple operations that can be analytically evaluated for randomness.
2. *A single test, such as the chi-square test, is sufficient to test the goodness of a random-number generator.* The sequence 0,1,2,...,$m$-1 will pass the chi-square test with a perfect score, but will fail the run test ⇒ Use as many tests as possible.
3. *Random numbers are unpredictable*. Easy to compute the parameters, $a$, $c$, and $m$ from a few numbers ⇒ LCGs are unsuitable for cryptographic applications

# Myths (Cont)

4. *Some seeds are better than others*. May be true for some.

$$x_n = (9806 x_{n-1} + 1) \bmod (2^{17} - 1)$$

- Works correctly for all seeds except $x_0 = 37911$
- Stuck at $x_n = 37911$ forever
- Such generators should be avoided.
- Any *nonzero* seed in the valid range should produce an equally good sequence.
- For some, the seed should be odd.
- Generators whose period or randomness depends upon the seed should not be used, since an unsuspecting user may not remember to follow all the guidelines.

# Myths (Cont)

5. *Accurate implementation is not important*.

- RNGs must be implemented without any overflow or truncation For example,

  $x_n = 1103515245 x_{n-1} + 12345 \bmod 2^{31}$

- In FORTRAN:

$x_n = (1103515245 x_{n-1} + 12345).AND.X'7FFFFFFF'$

- The AND operation is used to clear the sign bit
- Straightforward multiplication above will produce overflow.

6. *Bits of successive words generated by a random-number generator are equally randomly distributed*.

- If an algorithm produces *l*-bit wide random numbers, the randomness is guaranteed only when all *l* bits are used to form successive random numbers.

# Example 26.7

$$x_n = (25173 x_{n-1} + 13849) \bmod 2^{16}$$

Notice that:

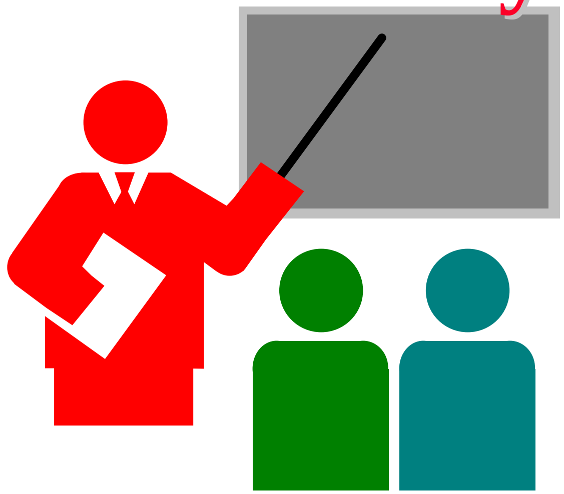
a) Bit 1 (the least significant bit) is always 1.
b) Bit 2 is always 0.
c) Bit 3 alternates between 1 and 0, thus, it has a cycle of length 2.
d) Bit 4 follows a cycle (0110) of length 4.
e) Bit 5 follows a cycle (11010010) of length 8.

| $n$ | $x_n$ | |
|---|---|---|
| | Decimal | Binary |
| 1 | 25,173 | 01100010 01010101 |
| 2 | 12,345 | 00110000 00111001 |
| 3 | 54,509 | 11010100 11101101 |
| 4 | 27,825 | 01101100 10110001 |
| 5 | 55,493 | 11011000 11000101 |
| 6 | 25,449 | 01100011 01101001 |
| 7 | 13,277 | 00110011 11011101 |
| 8 | 53,857 | 11010010 01100001 |
| 9 | 64,565 | 11111100 00110101 |
| 10 | 1945 | 00000111 10011001 |
| 11 | 6093 | 00010111 11001101 |
| 12 | 24,849 | 01100001 00010001 |
| 13 | 48,293 | 10111100 10100101 |

# Example 26.7 (Cont)

- The least significant bit is either always 0 or always 1.
- The $l$th bit has a period at most $2^l$. ($l$=1 is the least significant bit)
- For all mixed LCGs with $m=2^k$:
  - The $l$th bit has a period at most $2^l$.
  - In general, the high-order bits are more randomly distributed than the low-order bits.
    $\Rightarrow$ Better to take the high-order $l$ bits than the low-order $l$ bits.

# Summary



- Pseudo-random numbers are used in simulation for repeatability, non-overlapping sequences, long cycle
- It is important to implement PRNGs in integer arithmetic without overflow => Schrage's method
- For multi-stream simulations, it is important to select seeds that result in non-overlapping sequences
- Two or more generators can be combined for longer cycles
- Bits of random numbers may not be random

# Homework

- Submit answer to Exercise 26.5. Submit code and report $x_{20,000}$

# Exercise 26.5

Implement the following LCG using Schrage's method to avoid overflow:

$$x_n = 40014 x_{n-1} \bmod 2147483563$$

Using a seed of $x_0$=1, determine $x_{10000}$.