IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- RANDOM NUMBER GENERATORS
- Programmers have the option of using seven different random number
- generators for "PASSGEN( ) 'S" (i.e., ?GENBITS, ?GENRNG, ?GENCHAR, ?GENDEC, and ?GENFLOAT); And four different generators for ?GENSEED.
- Below is the rationale for which to choose.
- TERMINOLOGY:
- SEEDGEN= Random number generator used for ?GENSEED (i.e.,the
- "seed generator" used as the ?GENSEED ALGORITHM)
- PASSGEN= Random number generator used for ?GENBITS,?GENRNG,
- ?GENDEC,?GENCHAR, AND ?GENFLOAT. (i.e.,the "pass
- generator")
- LCG= Linear Congruent Generator
- CLCG= Combined Linear Congruent Generator
- LFG= Lagged Fibonacci Generator
- A= Forward multiplier for LCG's
- B= Backward multiplier for LCG's
- C= Additive constant for LCG'S

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- X{I}= Present seed
- X{I-1}= Previous seed
- Q= Special "decomposition" variable for LCG's
- R= Special "decomposition" variable for LCG's
- M= Modulus
- M_CLCG= Modulus for CLCG
- J= Lag for LFG'S (the longer one)
- K= Lag for LFG'S
- X{I-J}= Previous {I-J} seed from LFG seed array
- X{I-K}= Previous {I-K} seed from LFG seed array
- OPERTR= The arithematic operator used for the LFG (+,OR *)
- PERIOD= How many numbers generated before sequence
- repeats (i.e.,the cycle-length)

- LINEAR CONGRUENT GENERATORS (LCG)
- *********************************
- LCG's are designated as LCG(A,C,M), and have a period
- equal to M, M/2, M/4, or M/8. LCG's have the form:
- X{I} = (((A)*X{I-1})+C)//M FOR FORWARD STEPPING
- X{I-1} = (((B)*X{I}) +C)//M FOR BACKWARD STEPPING
- However, the intermediate products A*X{} and B*X{} must
- be kept from creating 32-bit overflow (unless M=2**32
- where the //M can just be ignored). If overflow can't be
- prevented, 64-bit simulated arithmetic must be used to
- include the overflow. To prevent overflow, a "decomposed"
- form of the above equation (if possible) must be used:

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

```
FORWARD:
  Q=M/A
  R=M//A
  IF    (A*(X{I-1}//Q) - A*(X{I-1}/R)) > 0
   X{I}= A*(X{I-1}//Q) - A*(X{I-1}/R)
  ELSE
   X{I}=(A*(X{I-1}//Q) - A*(X{I-1}/R))+M
BACKWARD:
  Q=M/B
  R=M//B
  IF      (B*(X{I}//Q) - B*(X{I}/R)) > 0
   X{I-1}= B*(X{I}//Q) - B*(X{I}/R)
  ELSE
   X{I-1}=(B*(X{I}//Q) - B*(X{I}/R))+M
```

But this only works if Q > R which is rare (for example, only 23,000 of the 4,000,000,000 32-bit LCG multipliers satisfy this. And finding a LCG with both backward and forward multipliers that satisfy this seems unlikely.

- COMBINED LINEAR CONGRUENT GENERATORS (CLCG)
- ***
- CLCG'S are made from two LCG's and have a period
- of (M1-1)*(M2-1)/2. They have the form:
- X{I} = ((LCG(A1,C1,M1)+
- (LCG(A2,C2,M2))//M_CLCG FORWARD
- X{I-1} = ((LCG(B1,C1,M1)+
- (LCG(B2,C2,M2))//M_CLCG BACKWARD
-

- LAGGED FIBONACCI GENERATORS (LFG)
- ********************************
- LFG'S are designated as LFG(J,K,M,OPERTR), and have a
- period of:
- ((2**J)-1)*(2**(LOG2(M)-1)) for the + operator
- and ((2**J)-1)*(2**(LOG2(M)-3)) for the * operator
- LCG's have the form:
- X{I} = (X{I-J} OPRERTR X{I-K})//M   FORWARD ONLY
- NO BACKWARDS YET

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- GOOD GENERATOR PROPERTIES ARE:
- A) Has been used for at least several years in industry or academia (i.e., well tested over time).
- B) Produces a string of numbers which approximates an independent and identically distributed source (I.I.D.). Independent means the probability of a number being generated is independent of when others generated (i.e., no conditional dependence). Identically distributed means all numbers have an equal probability of being generated (i.e.,a uniform distribution). For independence, rely on documented testing in the published literature. For identically distributed, additional testing was done to examine the bit uniformity of each generated 32-bit word for each generator; only the "best" bits are used when using a generator as a pass generator. The entire word is used when using a generator as a seed generator.
- C) Long periods (i.e.,want many numbers to be produced before generator starts over). "Cycle" and "period" are synonymous.

- D) Can generate non-overlapping segments. Each SAK program pass
- causes a string (a segment) of numbers to be generated by the
- pass generator (assuming the program contains some PASSGEN( ) 's).
- Non-overlapping segments means no significant part of any
- two segments will be identical. The generator's period can be
- broken into non-overlapping segments.
- This is only possible using the "FIBP" generator. However,
- any generator with a large enough period will most likely
- produce mostly non-overlapping segments for a typical set of
- SAK program passes. For example, a program with 100 PASSGEN( ) 's
- will use a segment of maybe 500 numbers; and if
- you run the program 100,000 passes, you have a total of
- 50,000,000 numbers used. Even generators with relatively
- small periods of 500,000,000 would use only 10% of their
- period for this example. There would be some overlapping
- segments -- but maybe not an undesirable amount. This example
- assumes relatively small PASSGEN( ) target lengths -- large
- ?GENBITS targets could lead to many over-lapping segments,
- but this might be acceptable for some applications.

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- E) Execution speed (both to startup and to run). SAK programs with
- many PASSGEN( ) 's or long PASSGEN( ) targets are referred to as
- "LONG RUNS" below. Some generators are not well suited for
- "SHORT RUNS" because of high initialization costs.
- F) May want no repeats of a number within a seed generator's
- cycle (i.e.,period) since a repeating base seed means
- an identical pass is generated (however, since preceding
- and following passes are most likely different, a different
- machine state may be tested). Repeating numbers are ok
- for pass generators -- only repeating sequences need to be
- avoided.
- G) Minimal seed memory requirements (i.e., more seeds means more
- overhead and record keeping).
- H) Minimal restrictions on initial seed.
- I) Reversibility. The seed generator for SAK must go backwards;
- and the pass generator used by the PASSGEN( ) 'S is sometimes
- desired to go backwards.

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- J) Repeatability. This is required for debugging. All of the
- generators below provide repeatability (both individually
- and when combined).
- The following seed and pass generators can be specified when
- using ?GENSEED, ?GENBITS, ?GENRNG, ?GENCHAR, ?GENDEC, or
- ?GENFLOAT.
- (#1)to(#4) can be used as either a seed or pass generator.
- (#5)to(#7) can only be used for a pass generator since they are
- not yet reversible.
- "MINSTD"(#4) is the default seed generator used by ?GENSEED.
- "RANDU" (#2) is the default pass generator used by ?GENBITS,
- ?GENRNG, ?GENDEC, ?GENCHAR, and ?GENFLOAT (i.e.,THE PASSGEN( )'S)
- If (#1,#3,#4,#5,#6,or #7) is specified by ?GENSEED as the default
- pass generator, that will be the default for all PASSGEN( ) 'S. A
- PASSGEN( ) can however change the pass generator for one invocation.
- Generator qualities have been subjectively graded below
- from A+ TO F:

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- 1) "OGSD" (OLD GENSEED)
- ********************
- FORWARD DESIGNATION: NONE, IT'S HOME-MADE
- BACKWARD DESIGNATION: NONE, IT'S HOME-MADE
- IID(OF 32-BIT WORDS).....?
- UNIFORMITY(BITS USED FOR PASSGEN).8:15(1 BYTE)
- PERIOD...........................2**26
- OVERLAPPING SEGMENTS..............YES
- STARTUP SPEED....................A
- "SHORT" RUN SPEED................D
- "LONG" RUN SPEED.................D
- REPEATS NUMBER WITHIN PERIOD......NO
- NUMBER OF SEEDS..................1
- RESTRICTIONS ON INITIAL SEED......NOT(0, EVEN, DIVISIBLE BY 5)

- 2) "RANDU"
- *******
- FORWARD DESIGNATION: LCG(65539,0,2**32)
- BACKWARD DESIGNATION: LCG(477211307,0,2***32)
- IID(OF 32-BIT WORDS).....D
- UNIFORMITY(BITS USED FOR PASSGEN).8:15(1 BYTE)
- PERIOD............................2**29
- OVERLAPPING SEGMENTS..............YES
- STARTUP SPEED.....................A+
- "SHORT" RUN SPEED.................A+
- "LONG" RUN SPEED..................A+
- REPEATS NUMBER WITHIN PERIOD......NO
- NUMBER OF SEEDS...................1
- RESTRICTIONS ON INITIAL SEED......NOT 0 OR EVEN

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

- NOTES:
- Derived in the 1970's by someone in SAK to be reversible
- and not create overflow. It was the SAK seed generator for
- 25 years. It has the following non-standard form:
- FORWARD:
- IF X{I-1}//2=0 THEN
- X{I}=X{I-1}+'124C41'X
- IF X{I-1}//5=0 THEN
- X{I}=X{I-1}+2
- X{I}=((X{I-1}//1000000000)*31627)//1000000000
- BACKWARD:
- IF X{I-1} IS EVEN THEN
- X{I}=X{I-1}+'124C41'X
- IF X{I-1}//5=0 THEN
- X{I}=X{I-1}+2
- X{I}=((X{I-1}//1000000000)*43222563)//1000000000

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

3) "IMPRV" (AN IMPROVED RANDU-TYPE GENERATOR)

- **
- FORWARD DESIGNATION: LCG(71365,0,2**32)
- BACKWARD DESIGNATION: LCG(814217229,0,2***32)
- IID(OF 32-BIT WORDS).....B-
- UNIFORMITY(BITS USED FOR PASSGEN).8:15(1 BYTE)
- PERIOD...........................2**29
- OVERLAPPING SEGMENTS..............YES
- STARTUP SPEED....................A+
- "SHORT" RUN SPEED................A+
- "LONG" RUN SPEED.................A+
- REPEATS NUMBER WITHIN PERIOD......NO
- NUMBER OF SEEDS..................1
- RESTRICTIONS ON INITIAL SEED......NOT 0 OR EVEN

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

4) "MINSTD" ("MINIMUM-STANDARD" VER. #2)

- ************************************
- FORWARD DESIGNATION: LCG(48271,0,(2**31-1))
- BACKWARD DESIGNATION: LCG(1899818559,(2***31-1))
- IID(OF 32-BIT WORDS).....B
- UNIFORMITY(BITS USED FOR PASSGEN).8:31(3 BYTES)
- PERIOD...........................2**31
- OVERLAPPING SEGMENTS..............YES
- STARTUP SPEED....................A
- "SHORT" RUN SPEED................B
- "LONG" RUN SPEED.................B
- REPEATS NUMBER WITHIN PERIOD......NO
- NUMBER OF SEEDS..................1
- RESTRICTIONS ON INITIAL SEED......NOT 0

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

NOTES:

- FORWARD:
- Using decomposed form to prevent 32-bit overflow with:
- Q=44488,R=3399
- BACKWARD:
- Using simulated 64-bit arithmetic to handle 32-bit overflow
- since Q^>R for reverse multiplier.
- This generator is the default SAK pass generator.
- Minimum Standard versions #2 and #3 are more random than
- version #1. Version #1 is the original Minimum Standard from
- the 1960's. All three versions are in "GENTAB COPY" (with
- backwards multipliers).

5) "CLCG" (COMBINES TWO LCG'S)

- ***************************
- FORWARD DESIGNATION #1: LCG(40014,0,2147483563)
- BACKWARD DESIGNATION #1: LCG(2082061899,2147483563)
- FORWARD DESIGNATION #2: LCG(40692,0,2147483399)
- BACKWARD DESIGNATION #2: LCG(1481316021,2147483399)
- IID(OF 32-BIT WORDS).....B+
- UNIFORMITY(BITS USED FOR PASSGEN).8:31(3 BYTES)
- PERIOD...........................2**63
- OVERLAPPING SEGMENTS.............YES
- STARTUP SPEED....................A
- "SHORT" RUN SPEED................B-
- "LONG" RUN SPEED.................B-
- REPEATS NUMBER WITHIN PERIOD......YES
- NUMBER OF SEEDS..................2
- RESTRICTIONS ON INITIAL SEED......NOT 0
-

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

NOTES:

- FORWARD:
- Using decomposed form to prevent 32-bit overflow with:
- Q1=53668,R1=12211 Q2=527744,R2=3791
- BACKWARD:
- Using simulated 64-bit arithmetic to handle 32-bit overflow
- since Q^>R for reverse multiplier.
- During initialization, the base seed created by ?GENSEED
- is used as the initial seed for both constituent generators.
-

6) "FIBM" (LAGGED FIBONACCI USING MULTIPLICATION)

- **
- FORWARD DESIGNATION: LFG(55,24,2**32,+)
- BACKWARD DESIGNATION: NOT YET DERIVED
- IID(OF 32-BIT WORDS).....A+
- UNIFORMITY(BITS USED FOR PASSGEN).7:30(3 BYTES)
- PERIOD...........................2**83
- OVERLAPPING SEGMENTS..............YES
- STARTUP SPEED....................C+
- "SHORT" RUN SPEED................B-
- "LONG" RUN SPEED.................A-
- REPEATS NUMBER WITHIN PERIOD......YES
- NUMBER OF SEEDS..................55
- RESTRICTIONS ON INITIAL SEEDS.....SEE NOTES

NOTES:

- Two seeds must be updated in the seed table each PASSGEN( )
- invocation, and the seed table must be initialized for
- each pass. The initialization requires filling the seed
- table with random values using another generator, then
- make all entries odd.

7) "FIBP" (LAGGED FIBONACCI USING ADDITION)

- **
- FORWARD DESIGNATION: LFG(521,168,2**32,+)
- BACKWARD DESIGNATION: NOT YET DERIVED
- IID(OF 32-BIT WORDS).....A
- UNIFORMITY(BITS USED FOR PASSGEN).7:30(3 BYTES)
- PERIOD...........................2**531
- OVERLAPPING SEGMENTS..............NO
- STARTUP SPEED....................D
- "SHORT" RUN SPEED................C+
- "LONG" RUN SPEED.................A-
- REPEATS NUMBER WITHIN PERIOD......YES
- NUMBER OF SEEDS..................521
- RESTRICTIONS ON INITIAL SEEDS.....SEE NOTES

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

NOTES:

- Two seeds must be updated in the seed table each PASSGEN( )
- invocation, and the seed table must be initialized for
- each pass. The initialization requires filling the seed
- table with random values using another generator, then
- to get a unique non-overlapping segment of the
- generator's cycle (i.e., to get the most uncorrelated
- program passes), the initial array must also be put into
- a "CANONICAL FORM". This is only possible for certain J,K
- pairs and is made by shifting left (zero into the LSB),
- clear the sign bit, then zero the entire last entry, then
- the LSB for one or two special entries is set on:

JK-PAIR	ENTRY
3,2	1
5,3	2,3
10,7	8
17,5	11
35,2	1
55,24	12
71,65	2
93,91	2,3
127,97	22
158,128	64
521,168	88 (THIS IS THE J,K PAIR CHOSEN FOR SAK by J. W.)

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

SUMMARY OF GENERATORS:

- **********************

	OGSD	RANDU	IMPRV	MINSTD	CLCG	FIBM	FIBP
WORD IID	?	D	B-	B	B+	A+	A
BITS USED (PASSGEN( ))	8:15	8:15	8:15	8:31	8:31	7:30	7:30
PERIOD	2**26	2**29	2**29	2**31	2**63	2**83	2**531
OVERLAPPING	Y	Y	Y	Y	Y	Y	N
STARTUP SPEED	A	A+	A+	A	A	C+	D
"SHORT" RUN SPEED	D	A+	A+	B	B-	B-	C+
"LONG" RUN SPEED	D	A+	A+	B	B-	A-	A-
REPEATS IN PERIOD	N	N	N	N	Y	Y	Y
NUMBER OF SEEDS	1	1	1	1	2	55	521
SEED RESTRICTIONS	MANY	>0,ODD	>0,ODD	>0	>0	MANY	MANY

"CONTROLLED RANDOMNESS"

- ************************
- The overall "RANDOM BACKBONE" of a succession of passes can be
- controlled through the selection of seed and pass generators.
- For example,
- For filling large data area's or
- for programs with few PASSGEN( ) 'S,
- Choose: SEEDGEN="MINSTD"
- PASSGEN="IMPRV"
- for very fast, reversible passes, a single seed, and
- ok randomness; but small period and overlapping segments.
- For programs with many PASSGEN( ) 'S (some reversible),
- Choose: SEEDGEN="MINSTD"
- PASSGEN="CLCG"
- for very random, reversible PASSGEN( ) 'S, and big period; but
- overlapping segments and two seeds to handle.

IBM S/390 RNG API User Manual by J. Wunderlich, PhD. 1997

For programs with many PASSGEN( ) 'S (none reversible),

- Choose: SEEDGEN="MINSTD"
- PASSGEN="FIBP"
- for the ultimate in non-correlated passes (i.e.,very good
- word independence and non-overlapping segments); but not
- reversible PASSGEN( ) 'S and 521 seeds.
-
- For any program where intentional lack of randomness and high
- correlation between passes is desired,
- Choose: SEEDGEN="OGSD"
- PASSGEN="OGSD"
- OR
- Choose: SEEDGEN="RANDU"
- PASSGEN="RANDU"
- This may closely simulate actual code execution (i.e.,lack
- of randomness and interdependence between passes may
- sometimes be a good thing!).

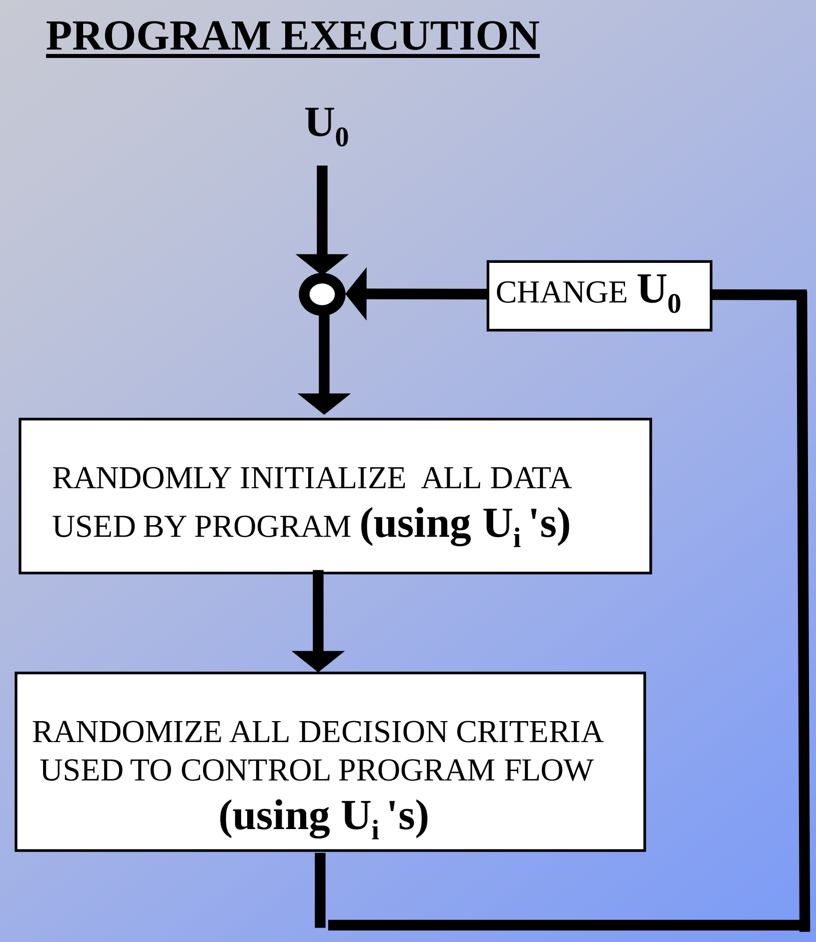
RANDOM NUMBER GENERATORS
FOR
ARCHITECTURAL VERIFICATION TEST-PROGRAMS

J. T. Wunderlich, Ph.D.

AGENDA

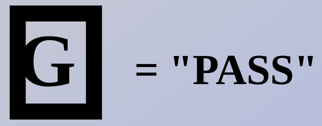
- PSEUDO-RANDOM PROGRAMS
- IDEAL GENERATOR
- GENERATOR TYPES
- GENERATOR QUALITY
- PARALLEL STRATEGIES
- SELECTED GENERATORS
- CONTROLLED RANDOMNESS
- IMPLEMENTATION

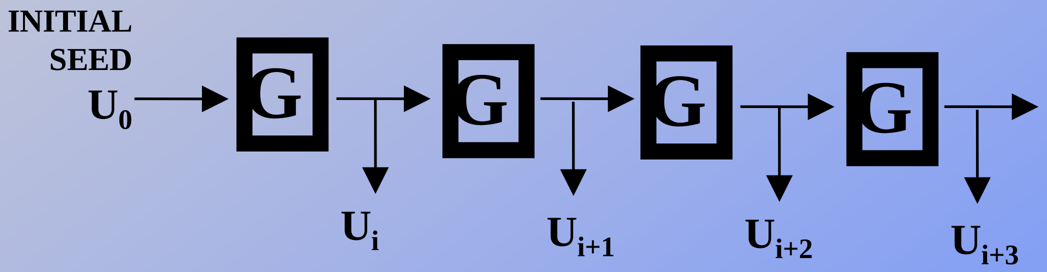
Controlled Randomness



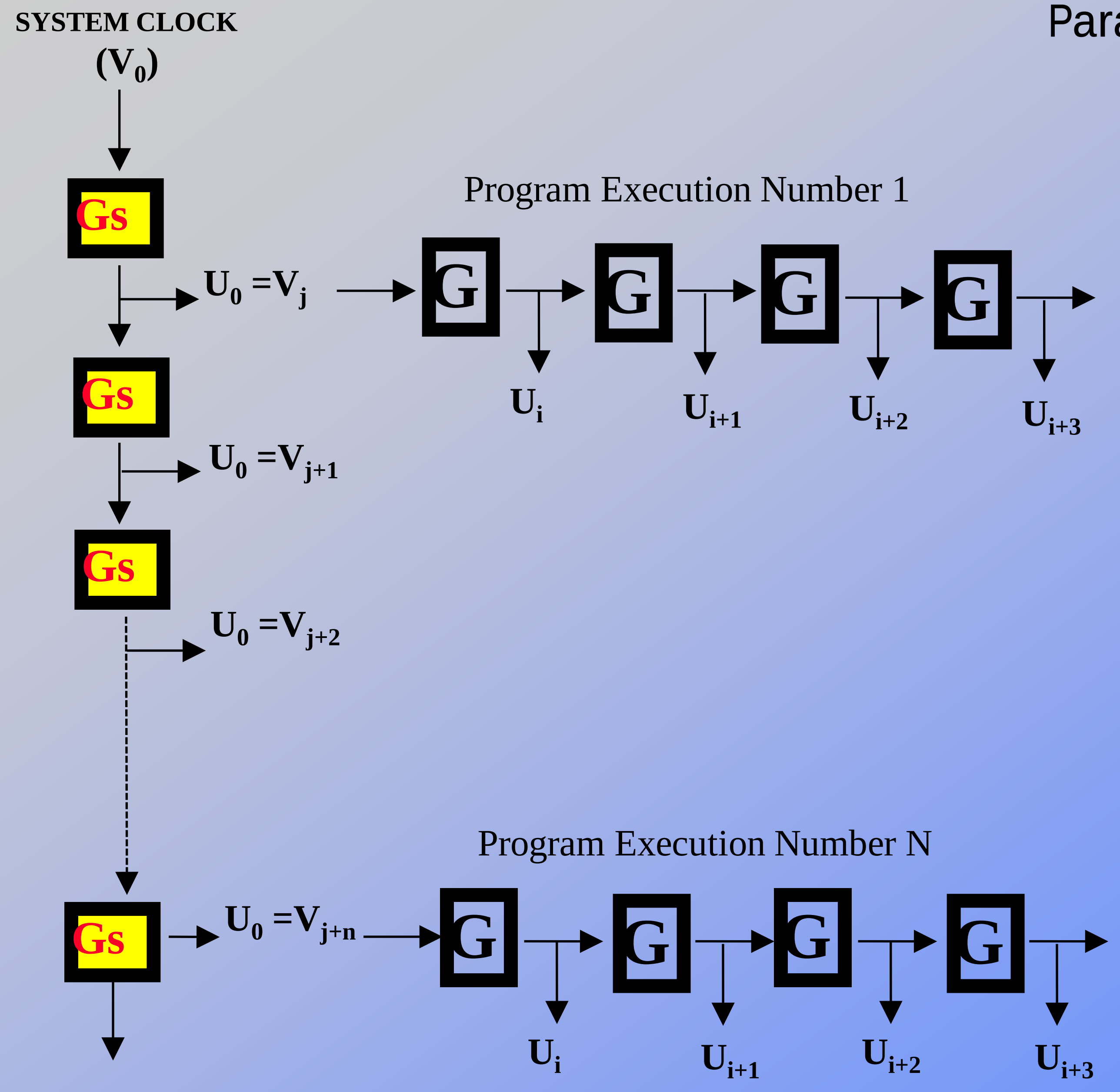
U_i = Randomly generated number

Controlled Randomness





Parallel Program Execution



Gs = SEED GENERATOR

G = PASS GENERATOR

Controlled Randomness

IDEAL GENERATOR

- **(IID) Independent AND Identically Distributed**
- **Identically Distributed: all numbers have equal probability of occurring**
- **Independent: probability of number being generated is independent of when other numbers generated. And therefore, P(A,B, . . . n) = P(A) * P(B) * . . . * P(n)**
- **LONG PERIOD (i.e., numbers generated before repeating)**
- **WELL TESTED**
- **FAST**
- **REPRODUCIBLE**
- **REVERSIBLE**
- **EASILY IMPLEMENTED (machine dependent)**
- **"SPLITTABLE"**

Controlled Randomness

TYPES OF GENERATORS

$U_i = [(a * U_{i-1}) + c] \bmod (m)$
where,
U_i = random number
U_0 = seed
a = multiplier
c = increment
m = modulus

- **LINEAR CONGRUENT (LCG's)**
  - **"Best analyzed"**
  - **"Most widely used"**

- **COMBINED LCG: stream pieced together from different generators. (LONG PERIODS)**

LCG's are a special case of $U_i = g(U_{i-1}, U_{i-2}, \ldots) \bmod (m)$
where the function $g(U_{i-1}, U_{i-2}, \ldots)$ **is** $(a * U_{i-1}) + c$
However, $g(U_{i-1}, U_{i-2}, \ldots)$ **can be:**

1) $a_1 U_{i-1} + a_2 U_{i-2} + \ldots\ldots a_n U_{i-n}$ **(LONG PERIODS)**

OR

2) $a_1 (U_{i-1})^2 + a_2 U_{i-1} + c$

OR

3) $U_{i-L} + U_{i-K}$ **(LONG PERIODS)**

Controlled Randomness

<u>TESTS OF HOW WELL GENERATOR APPROXIMATES AN IID SOURCE</u>

- **EMPIRICAL (Localized and limited)**
    - **Chi-Square, Serial, Run-up, etc.**
    - **Very Problem-Specific**

- **THEORETICAL (Global)**
    - **Evaluate geometric structure of scatter-plots:**
        - **Lattice tests (e.g., cubic lattice test)**
        - **Spectral test: Measure distance between hyperplanes**

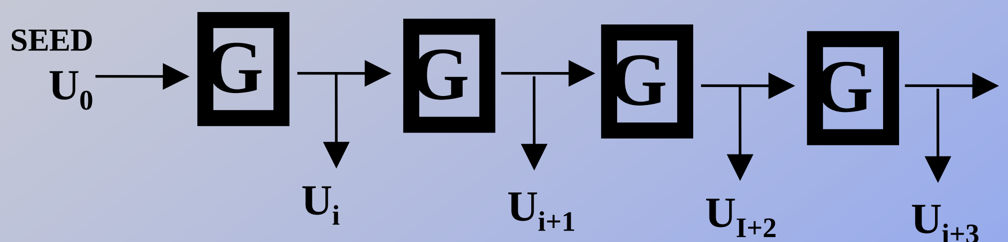
        - **Can't use for some types of generators**

Controlled Randomness

EXAMPLE GENERATOR AND SCATTER PLOTS FOR OVERLAPPING PAIRS

U(i) = generated number at time i

G = example generator: $U_i = [(a * U_{i-1}) + 1] \bmod (64)$



a=37

image corrupted

a=21

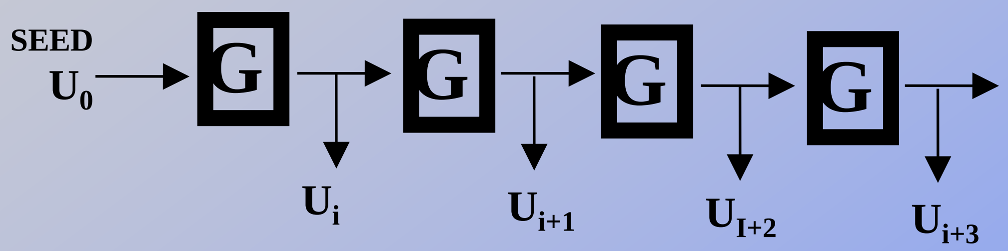
image corrupted

Controlled Randomness

EXAMPLE GENERATOR AND SCATTER PLOTS FOR OVERLAPPING TRIPLES

U(i) = generated number at time i

G = example generator: $U_i = [(a * U_{i-1}) + 1] \bmod (64)$



a=37

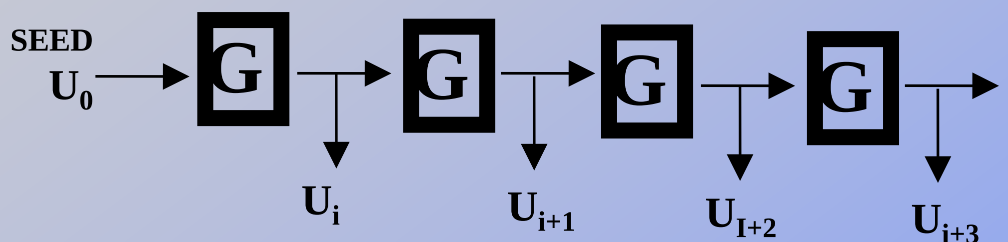
image corrupted

Controlled Randomness

GOOD AND BAD LCG'S

U(i) = generated number at time i

G = example generator: $U_i = [(a * U_{i-1}) + 1] \bmod (64)$



a=37

image corrupted

Controlled Randomness

GOOD AND BAD LCG'S

MINSTD (a good one)

$U_i = [(a * U_{i-1}) + 0] \bmod (2^{31} - 1)$ a ={ 16807,48271,69621}

good spectral tests, a full (m-1) period with no increment needed and no restrictions on U_0, well tested both empirically and theoretically, 390 assembler code exists.

SPECTRAL MEASURES:

	a = 16807	a = 48271	a = 69621
S = 2	**0.3375**	0.8960	0.7836
3	0.4412	0.8269	0.9205
4	0.5752	0.8506	0.8516
5	0.7361	0.7332	0.7318
6	0.6454	0.8078	0.7667
7	0.5711	0.5865	0.6628
8	0.6096	0.4364	0.7845

SCATTER PLOT OF OVERLAPPING PAIRS ……. *image corrupted*

--

S = Dimension of tuple-space (i.e., number of successive U_i's)

SPECTRAL MEASURE = Theoretical max / Max distance between hyperplanes

GOOD AND BAD LCG'S

RANDU(a bad one)

$U_i = [(65539 * U_{i-1}) + 0] \bmod (2^{31})$

S	SPECTRAL MEASURE
2	0.931
3	**0.0119**
4	0.0594
5	0.157
6	0.293
7	0.453
8	0.617

SCATTER PLOT OF OVERLAPPING TRIPLES …….. *image corrupted*

S = Dimension of tuple-space (i.e., number of successive U_i 's)

SPECTRAL MEASURE = Theoretical max / Max distance between hyperplanes

Controlled Randomness

GOOD AND BAD LCG'S

MINSTD

$U_i = [(a^* U_{i-1}) + 0] \bmod (2^{31} - 1)$ a ={ 16807,48271,69621}

good spectral tests, a full (m-1) period with no increment needed and no restrictions on U_0, well tested both empirically and theoretically, 390 assembler code exists.

SPECTRAL MEASURES:

	a = 16807	a = 48271	a = 69621
S = 2	0.3375	0.8960	0.7836
3	0.4412	0.8269	0.9205
4	0.5752	0.8506	0.8516
5	0.7361	0.7332	0.7318
6	0.6454	0.8078	0.7667
7	0.5711	0.5865	0.6628
8	0.6096	0.4364	0.7845

SCATTER PLOT OF OVERLAPPING PAIRS ……. *image corrupted*

S = Dimension of tuple-space (i.e., number of successive $\mathbf{U_i}$'s)

SPECTRAL MEASURE = Theoretical max / Max distance between hyperplanes

GOOD AND BAD LCG'S

BCPL

$U_i = [(2147001325 * U_{i-1}) + 715136305] \bmod (2^{32})$

Good spectral tests, a full (m-1) period, but increment needed. No restrictions on U_0.

Not as well tested as MINSTD. Mod computation eliminated!

S	SPECTRAL MEASURE
2	0.91
3	0.85
4	0.88
5	0.78
6	0.55
7	0.60
8	0.65

SCATTER PLOT OF OVERLAPPING PAIRS …….. *image corrupted*

S = Dimension of tuple-space (i.e., number of successive $\mathbf{U_i}$'s)

SPECTRAL MEASURE = Theoretical max / Max distance between hyperplanes

Controlled Randomness

PARALLEL STRATEGIES

- **CHANGE GENERATOR EACH PASS**
  - **Limited number of generators**
- **CHANGE LCG MULTIPLIER EACH PASS**
  - **Limited number of multipliers**

OR

1) MAXIMIZE STREAM LENGTH (PERIOD)

- **PRIME MODULUS LCG**
- **64-BIT LCG**
- **COMBINED LCG**
- **OTHER GENERATORS**

$U_i = g(U_{i-1}, U_{i-2}, \ldots) \bmod (m)$

where $g(U_{i-1}, U_{i-2}, \ldots)$ is:

$a_1 U_{i-1} + a_2 U_{i-2} + \ldots\ldots\ldots a_n U_{i-n}$

OR

$U_{i-L} + U_{i-K}$ **(lagged fibonacci)**

2) SPLIT UP STREAM FOR PASSES

- **NO SEEDING PASSES, JUST WRAP**
  - **Need bookkeeping for initial seeds**
- **RANDOMLY CHANGE INITIAL SEED**
  - **Avoids alignment with bad lattice features**
  - **Overlapping stream segments**
    - **minimize with large period**
    - **avoid with canonical form**

Controlled Randomness

SEED GENERATOR vs. PASS GENERATOR

SEED GENERATOR

PERIOD : MAKES NUMBER OF DIFFERENT PASSES. SMALLER FOR MORE PASS CORRELATION.

RANDOMNESS: LESS IMPORTANT THAN FOR PASS GENERATOR. IF DIFFERENT THAN PASS GENERATOR, OVERLAP MINIMIZED.

SPEED: LESS IMPORTANT THAN FOR PASS GENERATOR.

REVERSIBILITY: NEEDED FOR DEBUGGING

PASS GENERATOR

PERIOD: IF EVENLY DIVISIBLE BY NUMBER OF PASS GENERATOR INVOCATIONS IN A PASS, FIRSTPASS WILL REPEAT WHEN PERIOD IS REACHED.

RANDOMNESS: CRITICAL FOR NO CORRELATION BETWEEN PASSES, AND WITHIN PASSES. NO OVERLAP YIELDS BEST RANDOMNESS.

SPEED: MOST IMPORTANT WHEN CREATING LARGE ARRAYS OF RANDOM DATA. INITIALIZATION TIME MORE COSTLY FOR SMALL PROGRAMS.

REVERSIBILITY: USED INFREQUENTLY

Controlled Randomness

SELECTED GENERATORS FOR IBM (by J. Wunderlich, 1997)

SEED GENERATORS

CODE NAME	NUMBER OF SEEDS	PERIOD	RANDOM QUALITY?	SPEED (initial/ running)	CAN GO BACKWARD
OLDGSEED	1	2^26	-	A/B	Y
LCGPRIME (DEFAULT)	1	2^31	B	A/B	Y

PASS GENERATORS

CODE NAME	NUMBER OF SEEDS	PERIOD	RANDOM QUALITY/ OVERLAP?	SPEED (initial/ running)	CAN GO BACKWARD
OLDLCG32 (DEFAULT)	1	2^29	D/Y	A+/A+	Y
NEWLCG32	1	2^29	B-/Y	A/A	Y
COMBOLCG	2	2^63	B+/Y	A-/B-	Y
FIBOMULT	55	2^83	A+/Y	C+/A-	N
FIBOPLUS	521	2^531	A/N	D/A	N

NOTE: ALL GENERATORS WELL TESTED(EXCEPT *OLDGSEED*)

NOTE: FOR "*CONTROLLED RANDOMNESS*", *OLDGSEED*, *LCGPRIME*, *OLDLCG32*, AND *NEWLCG32* CAN BE SPECIFIED AS BOTH SEED AND PASS GENERATORS

Controlled Randomness

<u>**API's developed by J. Wunderlich, 1997**</u>

EXAMPLE USE OF J. Wunderlich API's by System's level programmers:

SEED GENERATOR ("*LCGPRIME*"):

FORWARD: **Gs:** $V_i = [(48271 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

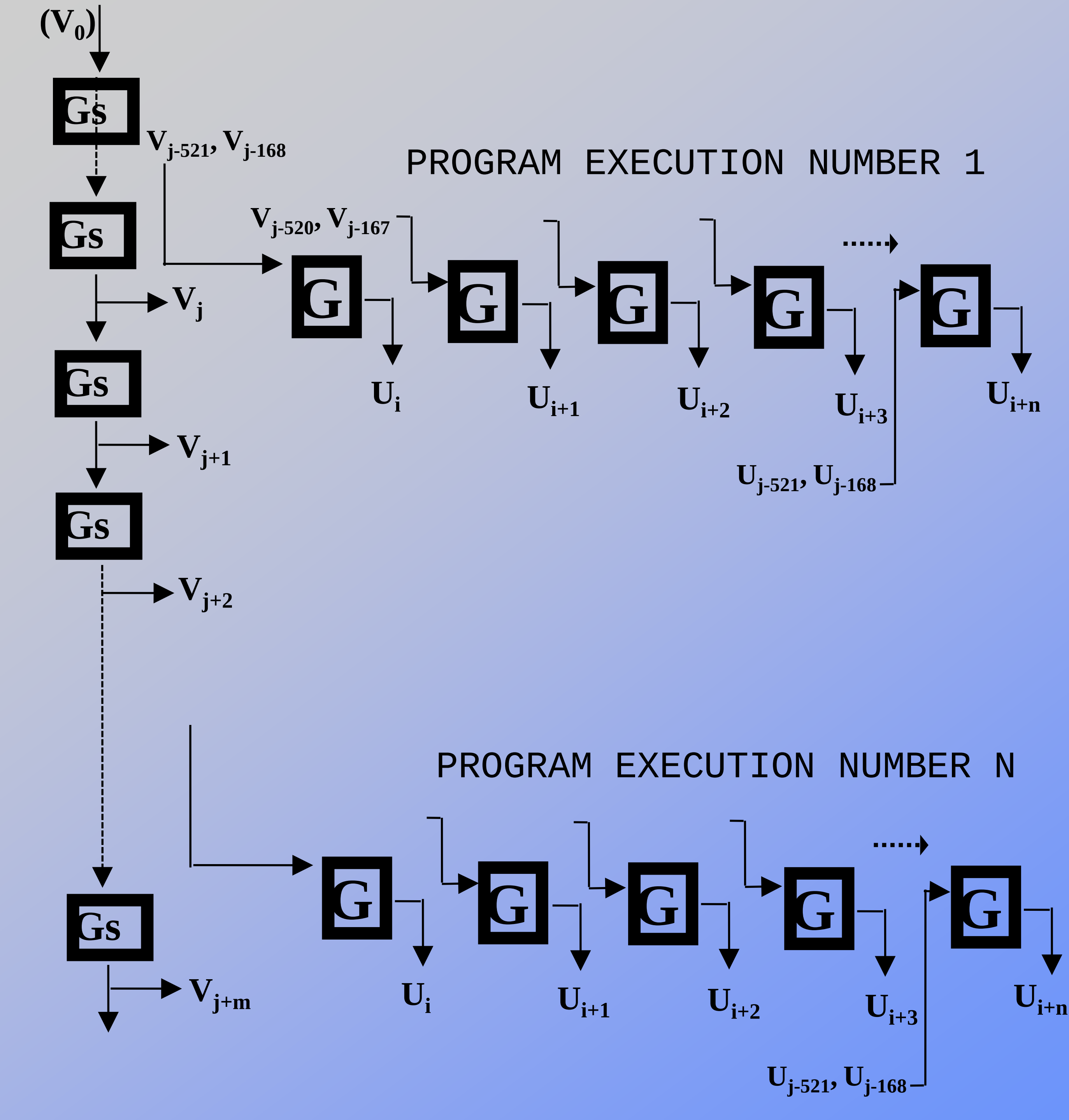
BACKWARD: **Gs:** $V_i = [(1899818559 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

PASS GENERATOR ("*FIBOPLUS*"):

G: $U_i = [U_{i-521} + U_{i-168}] \bmod (2^{32})$

API CODE SYNTAX:

```
?GENSEED[SEEDGEN(XSEEDGEN)][PASSGEN(XPASSGEN)]
PASSGEN( ) **........................[PASSGEN(XPASSGEN)]
where *** is BITS,CHAR,DEC,FLOAT,or RNG
```



API CODE EXAMPLE

SEED GENERATOR ("*LCGPRIME*"):

FORWARD:

Gs: $V_i = [(48271 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

BACKWARD:

Gs: $V_i = [(1899818559 * V_{i-1}) + 0] \bmod (2^{31} - 1)$

PASS GENERATOR ("*FIBOPLUS*"):

G: $U_i = [U_{i-521} + U_{i-168}] \bmod (2^{32})$