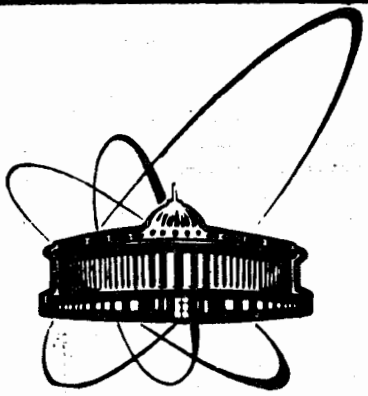


объединенный
институт
ядерных
исследований
дубна

Р11-89-461

Г.А.Ососков, Н.И.Чернов

ГЕНЕРАТОР С БОЛЬШОЙ ДЛИНОЙ ПЕРИОДА ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ ДЛЯ ПЕРСОНАЛЬНОЙ ЭВМ

Направлено в совместный научный сборник ОИЯИ-ЦИФИ "Алгоритмы и программы для решения задач физики"

1989

Ососков Г.А., Чернов Н.И. Р11-89-461

Генератор с большой длиной периода
псевдослучайных чисел для персональной ЭВМ

Предложена быстрая программа генератора случайных чисел, написанная на языке ассемблера для персональных компьютеров типа IBM PC/XT. Генератор использует мультипликативный алгоритм и выдает четыре случайные последовательности с большой длиной периода /от 2^{31} до 2^{50}/.

Работа выполнена в Лаборатории вычислительной техники и автоматизации ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна 1989

Перевод авторов

Ososkov G.A., Chernov N.I. P11-89-461

Random Number Generator with Prolonged
Period on a Personal Computer

A fast assembly language of a random number generator on personal computers IBM PC/XT-compatible is proposed. The generator uses the multiplicative algorithm and produces by options four random sequences with prolonged periods(from 2^{31} to 2^{50}).

The investigation has been performed at the Laboratory of Computing Techniques and Automation, JINR.

Preprint of the Joint Institute for Nuclear Research. Dubna 1989

В основе широко используемых методов машинной имитации случайных процессов, многих численных методов и, в частности, вычислительных экспериментов лежит генерирование на ЭВМ случайных чисел. Алгоритмам такой генерации посвящена обширная литература (см., например, [1-5]). Показано, что случайные числа с любым вероятностным распределением могут быть получены соответствующим неслучайным, детерминированным преобразованием из случайных чисел с равномерным распределением на отрезке $[0,1]$. Поэтому достаточно ограничиться проблемой генерации последних.

Как следует из указанной литературы, самый распространенный и единственно надежный способ получения случайных чисел на ЭВМ – это вычисление т.н. псевдослучайных чисел. Они генерируются по рекуррентным формулам, т.е. фактически являются детерминированными. Тем не менее большие наборы таких чисел обладают свойствами, позволяющими использовать их как последовательность независимых случайных чисел (например, они удовлетворяют статистическим тестам, предназначенным для проверки распределения и корреляции случайных величин).

Генераторы случайных чисел (ГСЧ) можно грубо разбить на эвристические, к ним, в частности, относятся так называемый метод перемешивания, т.е. манипулирование фрагментами машинной записи предыдущего случайного числа для получения последующего [4-7], и теоретико-числовые. В одном из самых подробных и квалифицированных обзоров по ГСЧ [1] достаточно убедительно показано преимущество теоретических алгоритмов, основанных на конгруэнтном методе Лемера. В этом методе n-е псевдослучайное число получается по формуле

$$X_n = A_n/R, \qquad (1)$$

где

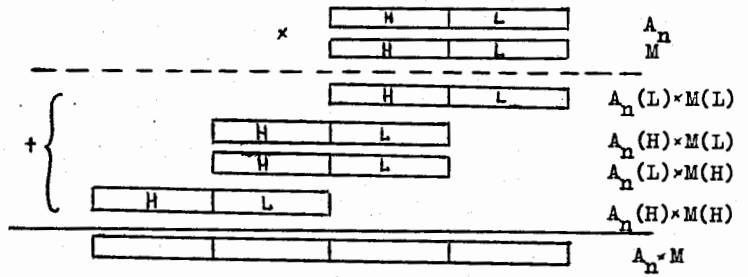
$$A_{n+1} = A_n \cdot M + C \pmod{R}. \qquad (2)$$

Здесь R, M, C, A_n $(n \geq 0)$ – целые числа. Константа R определяется разрядностью ЭВМ p (обычно $R = 2^p$). Величины M и C тщательно подбирают так, чтобы улучшить корреляционные свойства и увеличить период последовательности $\{X_n\}$ [1]. Отметим, что последовательность (2) всегда периодична с периодом $T \leq R$ (а в случае $R = 2^p$ и $C = 0$ период $T \leq R/4$). Обеспечение большого периода необходимо в

крупных машинных экспериментах, требующих многократного обращения к ГСЧ.

В монографиях /I,2/ рекомендуется более простой вариант метода Лемера: т.н. мультипликативный генератор, получающийся из (I-2) при $C = 0$. Утверждается /2/, что его свойства ничем не хуже, чем у метода Лемера ($C \neq 0$). Наконец, следуя рекомендациям работ /I,8/, мы выбираем множитель M в виде $M = 5^{2m+1}$, где m находится из условия $5^{2m+1} < R \leqslant 5^{2m+3}$. Период этого ГСЧ при $R = 2^P$ и $A_0 = 1$ максимален – он равен $R/4$.

Реализация мультипликативного ГСЧ на ЭВМ не представляет принципиальных трудностей ввиду простоты формул (I-2). Сложности возникают лишь при вычислении произведения $A_n \cdot M$ из (2), т.к. на малоразрядных персональных ЭВМ результат не умещается в машинное слово. Для решения этой проблемы применяется /8/ разбиение двоичной записи чисел A_n и M на $k \geqslant 2$ частей, которые затем попарно перемножаются и результаты складываются поразрядно. Общая схема при $k = 2$ приведена на следующем рисунке.



Заметим, что если $R = 2^P$, то достаточно вычислить младшие p битов произведения $A_n \cdot M$, что сокращает число операций в этой схеме.

Авторы работы /8/ реализовали этот ГСЧ на языке ассемблера на ЭВМ СМ-4 для четырех вариантов выбора M и R :

Таблица I

№ п/п	M	R
I	5^{17}	2^{40}
2	5^{19}	2^{48}
3	5^{21}	2^{52}
4	5^{23}	2^{56}

Выбор генератора №4 особенно удобен на СМ-4, т.к. мантисса вещественного числа с двойной точностью на языке ФОРТРАН имеет длину 56 бит. Генераторы I-4 работают на СМ-4 от 3-х до 6-ти мс на одно случайное число /8/.

Авторы настоящей работы воспользовались идеей алгоритма /8/ при создании ГСЧ для персональной ЭВМ типа IBM PC/XT(EC-I840,I832,I833, "Правец-I6"). Для получения максимальной скорости работы ГСЧ реализован на языке ассемблера. Одна версия программы написана в кодах микропроцессора INTEL 8088, другая – в кодах микропроцессоров 8088 и 8087. В следующей таблице приводятся подробные характеристики созданных программ:

Таблица 2

№№ п/п	M	R	Длина периода	Время (мс) на 8088	Время (мс) 8088/8087
I	5^{13}	$2^{31}-I$	$2^{31}-I$	0,5	0,3
2	5^{17}	2^{40}	2^{38}	0,5	0,45
3	5^{19}	2^{48}	2^{46}	0,5	0,45
4	5^{21}	2^{52}	2^{50}	0,55	0,5

Время работы ГСЧ на одно случайное число оказалось на порядок лучше, чем в /8/. Значение A_0, как и в /8/, для всех ГСЧ выбиралось равным I.

Генератор № 4 из табл.I не включен в табл.2, поскольку мантисса числа с двойной точностью в процессоре 8087 (т.е. в ФОРТРАНЕ для PC) имеет длину 52 бита, поэтому генератор № 3 из табл.I уже обеспечивает её максимальную "загрузку". Вместо этого мы включили в программу новый мультипликативный генератор – № I (табл.2). Он обладает одним важным преимуществом, а именно: у генераторов № I-4 из табл.I случайные числа имеют повторяющиеся младшие биты, см /I/. Это не сказывается в научных расчетах, но препятствует использованию случайных чисел в качестве случайной последовательности бит (например, для получения случайного адреса в памяти ЭВМ). У генератора № I из табл.2 все значащие биты (их 3I) равноправны, т.е. одинаково "случайны" и "независимы" (в указанном выше смысле). Это связано с тем, что число $R = 2^{31}-1$ простое (см. /I/, гл.3).

Генератор № 2 из табл.2 идентичен библиотечному генератору на 48-разрядной ЭВМ БЭСМ-6. Его статистические свойства высоко оцениваются в работе /2/.

Наша программа оформлена как подпрограмма-функция для вызова на языке ФОРТРАН. Вызов осуществляется командой **X=RANF(LEVEL)** . Функция **RANF** имеет тип **REAL*8** (или **DOUBLE PRECISION**). Параметр **LEVEL** целочисленный, принимает значения I,2,3,4 и задает номер генератора из табл.2. Каждый из ГСЧ № I-4 представляет собой независимую программу, использующую раздельные области памяти. Предусмотрена также возможность возобновления псевдослучайной последовательности с любого её члена. Для этого пользователь должен сохранить последнее полученное случайное число X_n. Затем для получения последовательности X_{n+1}, X_{n+2} , ... достаточно предварительно вызвать подпрограмму **RANFIN**, включенную в ГСЧ:

CALL RANFIN (LEVEL,X) .

Здесь **LEVEL**, как и выше, задает номер генератора, а **X** есть **n**-е случайное число X_n. Подпрограмма **RANFIN** подставляет значение $A_n = X_n R$ вместо "семени" A_0 (см.(I-2)).

Ниже приводится текст программы ГСЧ на языке ассемблера для микропроцессоров 8088/8087. В данном варианте передача параметров осуществляется так, как принято в ФОРТРАНЕ версии **Microsoft** (см.Информационный бюллетень ЛВТА, № 33).

В заключение отметим, что в появившихся в последнее время задачах теоретической физики, связанных с расчетами на решетках в квантовой хромодинамике, требуются ГСЧ с длиной периода, большей 2^{50}. Для таких задач описанный ГСЧ непригоден. Авторы могут рекомендовать различные методы увеличения периода (см., например, /7/). Можно использовать идею Г.Марсальи /9/ о спаренных генераторах. Практически неограниченную длину периода имеют матричные генераторы, основанные на идеях работы Таусворта (см.,например, /10/). Однако их реализация на персональных ЭВМ затруднительна, т.к. по своей природе они ориентированы на ЭВМ параллельного действия.

Приложение

```
     .8087                    ;  coprocessor 8087 is used
_data     segment word public 'data'
m1        dd      1220703125  ; 5**13  (in decimal)
exp1      dd      7fffffffh   ; 2**31-1 (in hexadecimal)
a1        dd      1           ; seed number for level=1
;    the block of the base number 5**21 for level=4
m4l       dd      4d6e2ef5h   ; low double-word of 5**21
m4lex     dd      0
m4h       dd      1b1aeh      ; high double-word of 5**21
m4hex     dd      0
```



```
;    put here the same block m3l,m3lex,m3h,m3hex for level=3
;                            (use 5**19 instead of 5**21)
;    put here the same block m2l,m2lex,m2h,m2hex for level=2
;                            (use 5**17 instead of 5**21)
a4l       dd      1           ; the block of the seed number for level=4
a4h       dd      0
;  put here the same blocks a3l,a3h and a2l,a2h for level=3,2
aall      dd      ?           ; ten-byte field for copies of the seed numbers
aalh      dd      0
aale      dw      ?
x1        dd      ?           ; eight words for temporary data
x2        dd      ?
x3        dd      ?
x4        dd      ?
exp2      dw      40          ; exponent for level=2
exp3      dw      48          ; exponent for level=3
exp4      dw      52          ; exponent for level=4
_data     ends
const     segment  word public 'const'
const     ends
_bss      segment  word public 'bss'
_bss      ends
dgroup    group     const, _bss, _data
assume    cs: _code, ds: dgroup, ss: dgroup, es: dgroup
_code     segment byte public 'code'
     public  ranf
ranf proc     far
     push     bp                 ; save bp
     mov      bp,sp              ; set frame pointer bp
     push     si                 ; save si
     push     di                 ; save di
     les      di,[bp+8]          ; load address of level
     mov      dx,es:[di]         ; load level into dx
     cmp      dx,1               ; begin testing the level value
     jne      $1
; the main block for level = 1
     fild     dword ptr exp1     ; load exponent (2**31-1) into 8087
     fild     dword ptr m1       ; load m1 into 8087
     fimul    dword ptr a1       ; multiply m1 by a1
     fprem                       ; get partial remainder
     fist     dword ptr a1       ; store the new seed number in a1
     fdiv     st(0),st(1)        ; normalize
     mov      di,[bp+6]          ; get address of function in DI
     fstp     qword ptr [di]     ; return the result
     fstp     st(0)              ; kill exponent in 8087
     mov      ax,di              ; return address of the function
     mov      dx,ss              ; this is conventional
     fwait
     jmp      out                ; end of job
$1:  cmp      dx,4               ; go on testing the level
     je       $4
;         ...........   (and so on)
; the main block for level = 4
$4:  fild     qword ptr m4h      ; load mh into 8087
     mov      ax,word ptr a4l    ; begin copying al into aall
     mov      bx,word ptr a4l[2]
     fild     qword ptr m4l      ; load ml into 8087
     mov      word ptr aall,ax   ; copy low word
     mov      word ptr aall[2],bx; copy high word
     fild     qword ptr aall     ; load al (its copy aall) into 8087
     mov      si,[bp+6]          ; prepare result address in si
```

```
      fld     st(0)               ; get copy of a1 in 8087
      fmul    st(0),st(2)         ; multiply a1 by m1
      fld     st(0)               ; get copy of m1*a1 in 8087
      fstp    tbyte ptr aa11      ; store m1*a1 in aa11:aa1h:aa1e
      fwait                       ; (temporary real format is ised!)
    • cmp     aa1e,16446          ; test if the exponent reachs its maximum
      je      $5                  ; equal, then aa11:aa1h is a valid result
      fistp   qword ptr aa11      ; store another copy of m1*a1 in aa11:aa1h
      jmp     $6
$5:   fstp    st(0)               ; kill another copy of m1*a1 in 8087
$6:   fmul    st(0),st(2)         ; multiply mh by a1
      mov     ax,word ptr aa11    ; begin copying aa11 into a1
      mov     bx,word ptr aa11[2]
      fistp   qword ptr x1        ; store mh*a1 in x1:x2
      mov     word ptr a41,ax     ; copy low word
      mov     word ptr a41[2],bx  ; copy high word
      fild    dword ptr a4h       ; load ah into 8087
      fmul    st(0),st(1)         ; multiply ah by m1
      mov     bx,word ptr aa1h    ; begin adding aa1h to x1
      add     bx,word ptr x1      ; (low word in bx)
      mov     cx,word ptr aa1h[2]; remove this for level=2,3
      adc     cx,word ptr x1[2]   ; remove this for level=2,3
      xor     ch,ch ;remove this for level=3; replace ch by bh for level=2
      fistp   qword ptr x3        ; store m1*ah in x3:x4
      xor     ax,ax               ; begin clearing aa1h
      mov     word ptr aa1h,ax    ;  (low word)
      mov     word ptr aa1h[2],ax;  (high word)
      fstp    st(0)               ; kill m1 in 8087
      add     bx,word ptr x3      ; begin adding (aa1h+x1) and x3
; replace here bx by bl and "word ptr" by "byte ptr" for level=2
      adc     cl,byte ptr x3[2]   ; remove this for level=2,3
      fstp    st(0)               ; kill m2 in 8087
;                now chop off the high bytes of the result
;                and store the rest of it in ah
      and     cl,0fh              ; remove this for level=2,3
      mov     word ptr a4h,bx     ; (low word)
      mov     word ptr a4h[2],cx  ; remove this for level=2,3
;               pack real*8 number
      fild    qword ptr a41   ; load result in integer*8 form
      fstp    qword ptr [si]  ; store result in real*8 form
      mov     ax,si           ; return address of the result
      mov     dx,ss
      fwait

      sub     [si+6],340h     ; divide by 2**m (subtract a constant from
;the exponent);replace 340h by 300h for level=3 and by 280h for level=2
      jmp     out       ; end of job
; Put here the main blocks for level=3 and for level=2. They must be
; the same as for level=4 but replace "4" by "3" (resp. by "2") in the
; identificators and make other replacements marked in the listing.
out :pop      di        ; restore di
     pop      si        ; restore si
     mov      sp,bp     ; restore sp
     pop      bp        ; restore bp
     ret      6         ; return, pop 6 bytes
ranf     endp
;*************************************************************
;  This is the entry to set the new seed number - 8-byte data
;*************************************************************
     public     ranfin
ranfin  proc    far
     push     bp                ; save  bp
     mov      bp,sp             ; set frame pointer bp
```



```
     les      bx,[bp+6]          ; get address of "new seed"
     fld      qword ptr es:[bx]  ; load "new seed" into 8087
     les      bx,[bp+10]         ; get address of level
     mov      dx,es:[bx]         ; load level into dx
     cmp      dx,1               ; begin testing the level value
     jne      $7
; block for level=1
     fimul    dword ptr exp1     ;  multiply by exponent 2**31-1
     fistp    dword ptr a1       ;  renew seed number a1
     jmp      out2
$7:  cmp      dx,4               ; go on testing the level
;          ............. (and so on)
; block for level=4
     fild     word ptr exp4      ;  load exponent into 8087
     fxch     st(1)              ;  prepare scaling
     fscale                      ;  scaling
     fistp    qword ptr a41      ;  renew seed number a1:ah
     jmp      out1
;  ........ put here the blocks for level=3 and for level=2
;  they must be the same as for level=4 but replace "4" by "3"
;  (resp., by "2") in identificators
out1:fstp     st(0)     ;  kill the rest of data in 8087
out2:mov      sp,bp     ; restore sp
     fwait
     pop      bp     ; restore bp
     ret      8      ; return, pop 8 bytes
ranfin  endp
_code   ends
     end
```

ЛИТЕРАТУРА

I. Кнут Д. Искусство программирования для ЭВМ, т.2. Мир, М., 1977.

2. Ермаков С.М., Михайлов Г.А. Статистическое моделирование. Наука, М., 1982.

3. Полляк Ю.Г. Вероятностное моделирование на электронных вычислительных машинах. Советское радио, М., 1971.

4. Голенко Д.И. Моделирование и статистический анализ псевдослучайных чисел на ЭВМ. Наука, М., 1965.

5. Соболь И.М. Численные методы Монте-Карло. Наука, М., 1973.

6. Ососков Г.А. Вопросы радиоэлектроники, сер.XII, общетехн., вып.13, 1959, с.52-66.

7. Ососков Г.А. ОИЯИ, Р11-80-520, Дубна, 1980.

8. Дагман Э.Е., Дагман В.Э. Программирование, в.1, 1989, с.85-89.

9. MacLaren M., Marsaglia G. J.A.S.M., v.12, n.11, 1965, p.83.

10. Lewis T.G., Payne W.H. J.A.S.M., v.20, n.3, 1973, p.456-468.

Рукопись поступила в издательский отдел
21 июня 1989 года.