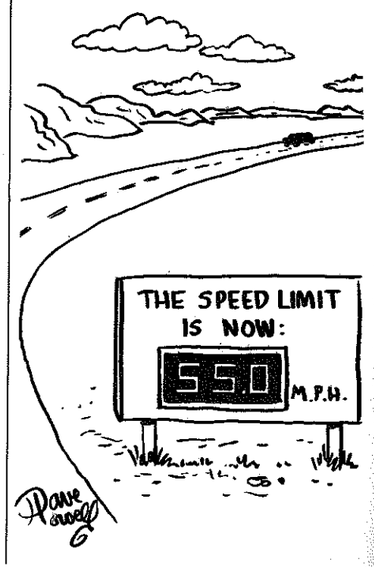
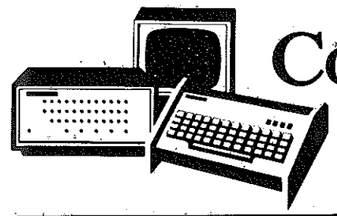
complaining about generally poor reception. This is largely due to increased lower-frequency absorption as the higher frequencies work better and better. It would be wise to be flexible enough to take advantage of the solar cycle, and DX each frequency range when it is best. Now is the time to concentrate on 20–60 MHz.

Tropical Bands Threatened. There's some apprehension that even when tropical reception improves toward the next solar cycle trough, DX listening may have been ruined on the 60-meter band. There have been proposals to allow high-powered international broadcasting on this band.

This would be very advantageous for the international listening audience when the MUF falls below 6 MHz, as it often does between Europe and North America during the winter in solar trough years. However, it would severely hinder reception of many third-world stations which presently occupy this band almost exclusively.

A letter-writing campaign seems to have had some success in opposing this move, coupled with the Third World's realization that they can bring to bear a powerful voting bloc at the World Administrative Radio Conference, set to begin this September in Geneva. The WARC will reallocate the entire electromagnetic spectrum, to conform with present-day needs and those anticipated until the end of the century. ◇





Computer Bits

By Hal Chamberlin

RANDOM NUMBER GENERATORS

RANDOM number generator subroutines are taken for granted by nearly all computer users. Every version of the BASIC programming language, even the simplest "tiny BASIC", has a random number function. Probably the heaviest usage of random numbers is in games where chance is an element. Other applications of random numbers include real-world simulations, program testing, and the generation of white noise in music programs.

While BASIC users have a seemingly infinite reservoir of random numbers at their disposal simply by using the RND function, machine-language users also have applications for random numbers. Serious mathematical simulations where BASIC is too slow or 6-digit accuracy is not enough, are written in machine language and may need a high-speed source of random numbers. Music synthesis is usually done in machine language and requires random numbers for noise generation and subtle variations in the sound for naturalness. Also, games on small computers such as the KIM-1 or any of the TV or arcade systems must be programmed in machine language, and they need random numbers as well. Thus we will be looking at random number generator programs in general and two in particular that are effective yet simple enough to program in machine language.

Properties. Although there is a variety of methods for generating random numbers, several traits are shared by all of them. The most distinctive is that the subroutine does not really *generate* random numbers, it merely *transforms* an input number into an output number. When generating a string of numbers, the previous output is simply fed back into the generator which then proceeds to generate a new output. The generator subroutine itself is a fixed mathematical function that does the input-to-output transformation. Although the output is related to the input in an obscure way, it seems to be completely unrelated to it in the end application.

The initial input used when the generator is started is called the *seed* and can usually be any number except zero. One consequence of the input-to-output transformation is that if the same seed is used on different occasions, the sequence of (pseudo) random numbers generated will be exactly the same. While this may not seem to be satisfactory random behavior, it does have advantages. For example, in debugging a program that utilizes random numbers, it is useful to be able to recreate known bugs and verify the effectiveness of corrections. On the other hand, when using a game program you want to be sure that the seed is different every time it is run. Forming a seed from the date and time supplied by the user is one possibility. Another is scanning through *all* of memory and forming the sum of what is found which is quite effective when semiconductor RAM is utilized for the memory function.

The generator transformation function

TABLE I—A AND B VALUES FOR LINEAR CONGRUENTIAL METHOD

Wordlength	Sequence length	A	B
8	256	77	55
12	4096	1485	865
16	65536	13709	13849
24	16777216	732573	3545443
32	4294967296	196314165	907633515

almost always uses integer arithmetic and the function is carefully chosen according to the word size of the computer. Since the output numbers are also integers with a finite number of bits, it is obvious that, at some point in the sequence, the seed will pop up again. From this point forward the sequence repeats itself.

An efficient random number generator will generate all or nearly all of the 2^n different numbers that can be represented by an n-bit word before repeating. Thus in a 16-bit computer, about 65,000 random numbers can be generated with single precision arithmetic before repeating.

The numbers produced by the generator are constrained to be within a certain *range*. The RND function in BASIC for example usually generates fractional numbers in the range of 0 to, but not including, +1.0. The machine language routines to be described generate unsigned integer numbers between 0 and 2^n-1. In either case, the range of the numbers is easily changed by multiplying and adding constants to them.

The random numbers themselves are *uniformly* distributed. This is a statistical term that means that all of the possible numbers in the allowable range are *equally* likely to occur. Some applications require *normally distributed* numbers where those near the middle of the range are more likely than those toward either extreme. This kind of distribution is most useful in simulating natural quantities such as the weight of a randomly chosen ripe apple. A true normal distribution can be closely approximated by adding up 12 uniform random numbers (range assumed to be between 0 and 1.0), and subtracting 6.0 from the sum. The mean (most likely value) of the normal distribution will then be zero and the standard deviation (a measure of how concentrated the numbers are around the mean) will be 1.0.

Methods. One of the most popular random number algorithms is called the *linear congruential* method. The transformation equation is: (new number)= [A × (old number)+B] MOD M where A and B are carefully chosen constants. Unsigned integer arithmetic is assumed when performing the multiplication by A and addition of B. The MOD function means to divide by M and keep just the remainder. If M-1 is the largest possible unsigned integer for the computer's wordlength, then the MOD function is automatically performed by *ignoring*

overflow when the other operations are performed; no actual division is necessary! Thus if the wordlength is 16 bits (one can combine several bytes to make any wordlength desired), multiplication by A can be performed with an integer multiply subroutine (see "Computer Bits" for Sept. 1978). Only the lower half of the product, which will almost surely overflow 16 bits, is added to B and the lower 16 bits of the sum, which may also overflow, is stored as the new random output.

Choosing A and B is tricky but the proper choice will insure that all 2^n (where n is the wordlength) possible numbers will be generated once before the sequence repeats. By using a wordlength of 32 bits, over 4 billion numbers can be generated before repeating. This is over a month running time at 1,000 numbers per second! If A and B meet the following requirements then not only will the sequence length be the maximum, but it will almost certainly be random enough for all but the most sophisticated applications.

1. A MOD 8 must be equal to 5.
2. A should be larger than $\sqrt{M}$ but smaller than M-$\sqrt{M}$.
3. A, when expressed in binary, should have a "random looking" bit pattern.
4. C should be an odd number.
5. C/M should be approximately equal to .21132.

Suitable values of A and B for some common wordlengths are in Table I.

One drawback of this type of generator, besides requiring a multiplication to perform, is that the least significant few bits of the numbers are not very random. Thus if random bits are necessary, the most significant bits should be used.

Another type of random number generator that is extremely efficient in machine language is based on simulating a feedback shift register in software. The only arithmetic operations needed are shifts and exclusive-ORs; no multiplication is required. Like the previous method, it is tied to the computer's wordlength but with multiple precision operations, any wordlength can be used. The procedure to generate a new number from an old one is as follows:

1. Shift the number left one bit position bringing in a zero on the right end and putting the overflow bit into the carry flag.
2. If the carry flag is off, the process is complete.
3. If the carry flag is on, *flip* selected bits in the shifted number. This may be

accomplished by exclusive-ORing a mask word with the shifted number. The process is now complete.

Thus the key to this generator is to find a suitable mask. It is highly desirable to use a mask that gives a maximum length sequence which in this case is 2^n-1 since zero transforms into itself. If such a mask is not used, some seeds may produce very short sequences, even for long words. The best way to find a suitable mask is to write a simple program that actually counts the number of iterations necessary before an initial seed of 0001 is returned. Suitable masks for several popular word-lengths are given in Table II.

The *numbers* produced by this generator are not as good as with the first generator but the individual *bits* produced are highly random. The results are more than adequate however for games and white noise generation and one could hardly ask for a simpler procedure. The performance as a number generator may be improved by iterating it several times (such as N+3 where N is the wordlength) to get each number.

TABLE II—MASK WORDS FOR FEEDBACK SHIFT REGISTER METHOD

Wordlength	Sequence length	Mask
8	255	1D
12	4095	1D9
16	65535	1D87
24	16777215	1D872B
32	4294967295	1D872B41

Testing. In any critical application it is necessary to test a homebrew random number generator before using it extensively. Even the RND function in BASIC is much better in some versions than others. While proper testing is a complex mathematical subject, one thing is definite: visually looking at a printout of a few numbers in the sequence is not a very reliable test. In fact, the "eyeball test" would probably flunk even the best generators because people tend to see patterns in small collections of things and any such patterns are not "supposed" to happen in a random sequence. One reasonably good visual test however is to generate a string of random bits and then fill the screen of a bit-mapped graphic display interface with them. Any visible regularity that covers a significant portion of the screen is probably a clue to a poor random number generator. ◇

Software Sources

8080/6800 File Management. "Now what was the name of that good French restaurant in Chicago?" If you filed it once, DATA FILE can find it. The program responds to any information you can recall about the contents of a file by displaying the rest of the file. It can, for example, remind you who belongs to all those long-distance numbers on your phone bill. The program is available for 6800 or 8080 systems, and runs in 1K, using remaining RAM for storage. Features include facilities for editing and updating files, automatic top-of-memory check ensuring no data is accidentally lost, and continuous display of memory addresses for saving all data on tape or disk. 6800 versions are available as a listing only, or with either K.C. cassette or SWTP disk, all starting at 0110 Hex. 8080 versions are available as listing only, with Tarbell or National Multiplex CC-7 or CC-8 cassette (specify) or Intel-format paper tape, or on North Star disk; starting address may be 0000, 2000 or 2A00 hex. Listings are $10, listing with cassette or tape $15, and listings with disk, $16. Practical Programming Co., Box 2069, North Brunswick, NJ 08902.

1802 Programming Aids. The *1802 Programmer's Notebook* includes: information on software relocation and register-assignment techniques, timing constants, clock calibration, and short programs for time-of-day clock, hex frequency counter, hex random number, and hex addition and multiplication. $1.00 plus self-addressed, stamped envelope, from David R. Wright, 128 Campus Ave., Ames, IA 50010.

PILOT for TRS-80. PILOT is a language designed for computer-assisted education, and is said to be so simple that even 6-year-olds have taught themselves to program in it. In this TRS-80 form, it includes PILOT program statements and commands (including CLOAD, CSAVE and line-printer commands) plus TRS-80 screen-clear and graphics commands. TRS-80 PILOT requires Level-II, 4K or 16K. $50. Jeff Lasman, PRACTICAL APPLICATIONS, P.O. Box 4139, Foster City, CA 94404.

TRS-80 Word Processor. The Electric Pencil word-processor program is now available for the Radio Shack TRS-80, as well as an optional serial-printer output interface with lower-case and control-key modification instructions. Written in machine code, not BASIC, for faster running, it will load into either Level-I or Level-II TRS-80s with 16K. It will operate upper-case only in unmodified machines, or upper/lower-case with the modification kit. Printers used can be either the Radio Shack printer and expansion box, or any RS-232 printer running 110-9600 baud, with the optional interface. Other features include 2-key rollover and repeat, line and character insert/delete, forward/reverse scrolling, string search with optional replace, block moves, page titling and numbering, and print formatting. The TRS-80 Electric Pencil is $100; the TRS-232 printer output interface with instructions only for lower-case modification (parts will be available later) is $40. Small System Software, Box 483, Newbury Park, CA 91320.

8080 Simulator for KIM-1. 8080 programs can be run on a 6502-based KIM-1 with this program. It executes the entire 8080 instruction set and maintains 8080 registers for convenient examination or modification. It runs in less than IK of memory, and can be relocated in ROM and adapted to other 6502 systems. $19.50. Dann McCreary, 4758 Mansfield St. #2P, San Diego, CA 92116.