Random Numbers in Machine Language for Commodore 64

COMPUTE! ISSUE 72 / MAY 1986 / PAGE 77

Random Numbers
In Machine Language
For Commodore 64

Neil Boyle

Here are two different methods for obtaining random numbers from machine language on the Commodore 64. Even if you're not a machine language programmer, you may find the explanation of the RND function useful in BASIC programming.

Sooner or later, nearly every programmer needs to generate random numbers. They're useful for introducing an element of uncertainty in games, as well as many other applications. Random numbers are easily available in BASIC with the RND function, but it's not so easy to generate these numbers in machine language. Two different methods are available on the Commodore 64: The first involves the BASIC RND function, and the second uses the SID (Sound Interface Device) chip.

BASIC's RND Function

The BASIC routine which performs RND is at memory location $E097 (decimal 57495) in the Commodore 64's Read Only Memory (ROM). This function generates values of different types, based on a seed number. Random numbers generated by the RND function are not truly random; rather, they are part of a very long, repeating number series (hence, the term pseudorandom numbers). However, they are usable as random numbers in most programs.

There are three different ways to use RND in BASIC. If you supply a positive number inside the parentheses-for instance, RND(1)-the first seed value used is the one copied from ROM to locations $8B-$8F when you turn the computer on. Each subsequent seed value is generated by scrambling the previous one. The RND function looks at the sign of the number in parentheses to see whether it's positive, but does not use the number itself. Thus, since RND(1234) and RND(1) both use positive values, both expressions produce exactly the same result.

What this means in practice is that RND with a positive number produces exactly the same number sequence each time you turn on the computer. To demonstrate, enter SYS 64738 to reset the computer, then enter the following line in direct mode (without a line number):

```
FOR J=1 TO 5:PRINT RND(1)NEXT
```

The computer prints these numbers:

```
.185564016
.0468986348
.827743801
.554749226
.897233831
```

Now reset the computer and enter the same line, substituting a different positive number in the parentheses. As you'll see, positive values generate the same results whether you use RND(1) or RND (65536). Since the reset stores the same values in the seed locations, and RND's scrambling method is predictable, the same numbers show up every time.

If you supply a zero inside the parentheses, RND gets values from the computer's internal timer A and the time-of-day clock on CIA chip # 1 to use as the seed. Since the clock values are constantly changing, this would seem to be a way to generate more truly random numbers. However, because the time-of-day clock operates with binary coded decimal numbers, certain values never appear in the seed. Thus, the randomness of RND(0) is questionable. To illustrate this, enter and run the following one-line program:

```
10 POKE 1024+(RND(0)*1000),160:
  GOTO 10
```

The many unfilled character spaces on the screen represent values that are never generated by RND(0).

When you supply a negative number in the parentheses, RND uses the value itself as the seed. For instance, enter the following line in direct mode:

```
X=RND(-654321):PRINT RND(1)
```

The result is always .333675369. If you substitute a different negative number, RND generates a different yet predictable result. This can be useful when testing a program: To generate the same series of numbers each time, set the seed with a negative number, then use RND with a positive number. In cases where you want a wholly unpredictable number series, the best method is to start with X=RND(-TI) to begin with an unknown seed value, then use positive RND arguments thereafter. Since the seed value depends on how many sixtieths of a second have elapsed since you turned on the computer (or reset the software clock with TI$), the results are unpredictable enough to satisfy most ordinary needs.

Calling RND From ML

To call the RND function from within an ML program, execute JSR $E09A. The value held in the accumulator register when you call the routine determines what RND does. If the accumulator holds a negative byte value ($80-$FF), then the value in floating point accumulator 1 (FAC1) is used as the seed. If the accumulator holds zero, RND uses values from timer A and the time-of-day clock, just as in BASIC. If the accumulator value is positive ($00-$7F), then the seed value in locations $8B-$8F is used.

Thus, your choices for ML programming are essentially the same as for BASIC, except that the system variable TI is not available as an argument. An alternative to using TI is to load the byte values from the software clock ($A0-A2) directly into the seed addresses ($8B-8F), thus giving a fairly random seed. If you'd rather not bother with loading the accumulator, you can perform JSR $E0BE to go directly to the routine which uses the stored seed value.

As you probably know from using RND in BASIC, the function always returns a floating point number between 0 and 1. In machine language, it is usually more convenient to use a single byte value in the range 0-255. Unfortunately, some of the randomness of the BASIC floating point number comes from scrambling the bytes in FAC1; this is lost when single bytes are used. One alternative is to convert the floating point number to an integer and use one or more bytes of the integer. But this is somewhat awkward. Nonrepeating random numbers involving all possible single byte values seem to be produced in locations $63 and $64 of FAC1 after a call to $E09A with the accumulator set appropriately. The values found in locations $62 and $65 do not include all the values from 0-255 and therefore should not be used.

SID's Random Number Generator

The 64's SID chip also has the ability to generate random values and is very easy to use. All you need to do is select the noise waveform for the SID's voice 3 oscillator and set voice 3's frequency to some nonzero value (the higher the frequency, the faster the random numbers are generated). It is not necessary to gate (turn on) the voice. Once this is done, random values appear in location $D41B. The parameters need only be set once, as shown in this example. (You'll need a machine language assembler to enter this and the following example; the semicolons and the comments which follow them are optional.)

```
LDA #$FF  ; maximum frequency value
STA $D40E ; voice 3 frequency low byte
STA $D40F ; voice 3 frequency high byte
LDA #$80  ; noise waveform, gate bit off
STA $D412 ; voice 3 control register
RTS
```

Once this code is executed, the SID chip continuously produces random byte values, which you can retrieve with a statement like LDA $D41B. This method works only on the Commodore 64 and 128, since only those two computers have a SID chip. Values obtained from the SID chip do seem to be random: Each of the values in the range 0-255 occurs at about the same frequency, and the series does not repeat in the first 34,000 values.

In many cases, you'll want to obtain random numbers only within a certain range. In BASIC, this is done with a statement like RND (1)*(U-L)+L. Here the variables U and L stand for the upper and lower limits, respectively, of the number range we want. This statement produces integer numbers within that range, excluding the first and last values in the range (for instance, if U=30 and L=1, then 30 and 1 do not appear in the series). To include the first and last values, use a statement like INT (RND(1)*(U-L+1)+L.

In machine language, we're usually interested in single byte values: integers in the range 0-255. Should larger values be needed, you can always generate two or more bytes and combine them. Getting numbers within a certain range is simply a matter of generating numbers until you find one that falls in the range you want. For instance, the following routine generates numbers within the range $10-$40 (before performing this routine, you must set up the SID chip as shown above):

```
RAND LDA $D41B  ; get random value
            from 0-255
    CMP #$31   ; compare to
             U-L+1
            ; U-L+1 = $31 =
            $40-$10+$01
    BCS RAND    ; branch if value >
            U-L+1
    ADC #$10   ; add L
    RTS
```

This routine generates random numbers until one is found that falls between 0 and the difference between the high and low values. Then the low value is added to the result to give a value between the low and high limits, inclusive. If the range is very small, many numbers may have to be generated before you find one that's suitable.

You can decrease the delay by ANDing the random number with a value that removes the unwanted higher bits. For instance, if the difference between the low and high limits is $0A, then AND the random value with $0F to remove the four high bits, then test whether it falls within the range. One useful special case deserves mention. To generate an integer in the range -1 to 1 inclusive, use the BASIC statement INT(RND(1)*3)-1. To get a value between $01 and $FF in machine language, AND the random value with $03 and use a routine like the one shown above, where L=$FF and U-L+1=$03.

The method you use to generate random numbers depends on your needs. In machine language, if repeatable values aren't needed, it's simplest to use the SID chip. Should repeatable values be required (for testing a routine, etc.), set the chosen seed in the seed locations $8B-$8F, call the ROM routine at $E0BE, then use the value found in locations $63 or $64 for the random number. Due to its questionable randomness, the timer/ clock method is not recommended.