COMPUTER TECHNOLOGY

Software timing for 6500 series microcomputers

JAMES M. PRICE

Oklahoma State University, Stillwater, Oklahoma 74074

Two machine language programs are presented for use in behavioral research requiring timing of reaction latencies, stimulus presentations, or intertrial intervals. Sample programs in BASIC show how to load and use the subroutines, and suggestions are made for possible modifications to a variety of situations.

Aside from routine data analysis, there are a number of ways in which today's microcomputers can be used in behavioral research. At a minimum, the video display allows the presentation of textual and graphic stimuli, and subjects can respond through the keyboard or game paddles. However, a great many experimental paradigms require accurate measurement of short time intervals, for example, stimulus presentations, reaction latencies, or intertrial intervals. A timer for such events should be able to resolve differences on the order of milliseconds.

Hardware clocks that meet the requirements of behavioral research are available for some computers and can be fabricated for others (e.g., Felsenstein, 1978; Goldband, 1979). All such timers involve some outlay of funds, and they may also involve modification of the computer itself. On the other hand, a very satisfactory software timing routine can be implemented at no cost to the experimenter other than a few minutes of programming time and a small amount of memory. The present article will illustrate two such routines originally developed for the APPLE II but adaptable to any computer using an MCS 6500 series microprocessor (e.g., Commodore PET, APF PeCos, and Ohio Scientific Challenger).

The basic operation involved in software timing is the execution of a "null loop," a sequence of commands that introduce a delay interval of known duration prior to incrementing the value stored in a counter. To measure reaction times, for instance, this loop is started at the time of stimulus presentation and allowed to run until the keyboard is activated, at which time the loop is terminated and the value in the counter is retrieved. For timing stimulus presentations, the loop is simply executed a specific number of times.

Rosner (1978) indicates a procedure that can be used to implement a null-loop timer within a BASIC program. A major problem with such methods is that the resolution of time intervals may be quite coarse. Because each BASIC statement must be interpreted into machine code at execution time, the delay built into the loop is a function not only of the execution of the statements, but also of their interpretation. Empirical tests of such

loops on the APPLE II indicate that 675 iterations of a BASIC loop are executed per second, yielding a resolution of .00148 sec/count. These tests were conducted without checking for keyboard activity, so actual resolution would be somewhat coarser.

A preferable method is to write the timing routine in machine language. Interpreter time is eliminated, thereby increasing the resolution of the timer. Furthermore, once stored in RAM, a machine code program can be called by any BASIC program subsequently loaded, provided power is maintained and the timer is not overwritten. Thus several programs, each requiring timing capabilities, can use the same subroutine, which can be stored separately on tape or disk. Hogenson (1977) demonstrates an assembler program that is a software analog of a hardware real-time clock for 8008-based computers. In one version, it serves as a delay timer, but extensive changes are needed in order to alter the resolution, since dummy commands are inserted or deleted to "fine tune" its measurements. If a software timer is needed at several points in a program, it must be capable of modification during execution, without operator intervention.

Figure 1 shows a timing subroutine written in both machine and assembler languages for the 6500 instruction set. (The instructions are the same, but the memory locations may vary from one machine to another.) The machine code may be loaded 1 byte at a time into the RAM locations indicated, or the assembler commands may be stored via assembler editing routines. In addition, the subroutine may be stored by "POKEing" the memory locations with the decimal equivalents of the machine code, as shown in Figure 2, using APPLE's version of BASIC.

The APPLE II computer reserves Memory Locations 768-1023 (300-3FF hex) for user-supplied machine language programs. Because the implementation routines that accompany the communications interface card are normally stored in Locations 768-939 (300-3AB hex), the timing subroutine is shown starting at Location 944 (3B0 hex). Once loaded, the subroutine will not interfere with the operation of the communications program,

Hex(Dec)	Assembly	Machine	Comments
Address	Language	Language	
380 (944)	LDA \$03DA	AD DA 03	load overflow flag
383 (947)	RNE \$03CD	00 18	if set, check keyboard
385 (949)	LDY\$03DB	AC DB 03	load delay factor
388 (952)	DEY	98	decrement delay factor
389 (953)	BNE \$03BB	00 FD	if non-zero, go back
38R (955)	INC \$03DC	EE DC 03	increment low counter
38E (958)	BEQ \$03C5	FD 05	if zero, go to high counter
3C0 (960)	NOP	5A	no operation
3C1 (961)	NOP	5A	
3C2 (962)	JMP \$03CB	4C CB 03	skip high-byte counter
3C5 (965)	INC \$03DD	EE DD 03	increment high counter
3C8 (968)	BNE \$03CD	00 03	if non-zero, check keyboard
3CA (970)	INC \$03DA	EE DA 03	set overflow flag
3CD (973)	LDX \$C000	AE 00 C0	load keyboard value
3D0 (976)	CPX #580	E0 80	compare with 128
3D2 (978)	BMI \$03B0	30 DC	if minus, start over
3D4 (980)	LDX #500	A2 00	load a zero
3D6 (982)	STX \$C010	8E 10 C0	store in keyboard strobe
3D9 (985)	RTS	60	return to calling program
3DA (986)			overflow flag
3DP (987)			delay factor
3DC (988)			low-byte counter
3DD (989)			high-byte counter

Figure 1. Machine code and assembly language listings of the timing subroutine.

```

10 DATA 173,218,3,208,24,172,219,3,136,208,253,238,220,3
20 DATA 240,5,234,234,76,200,3,238,221,3,208,3,238,218,3
30 DATA 174,0,192,224,128,48,220,162,0,142,16,192,96
40 FOR I = 944 TO 985
50 READ X
60 POKE I,X
70 NEXT I
80 END

```

Figure 2. BASIC program to load the timing subroutine into memory

and it will reside in a location not available for storage of BASIC programs or variable tables. These locations are used by the disk operating system at "boot-up," so the programs must be reloaded whenever the disk system is enabled. Appropriate memory locations for other computers may be found by consulting the applicable memory map.

Four memory locations are used by the subroutine, two containing the high and low hex bytes of the count, a third containing a delay factor, and a fourth as an overflow flag. Although the choice of these locations may be specified by the user, the illustrated program uses the four locations immediately following the program itself. Thus Location 987 (3DB hex) contains the delay factor, the number of times the null loop will be executed between successive updates of the counters. This delay is specified by the user and is discussed later in this paper. Locations 988 and 989 (3DC and

3DD hex) are the counter locations. The overflow flag is stored in 986 (3DA hex).

When the subroutine is called, Location 986 is checked to determine whether the overflow flag is set to a value of zero. If so, the user-supplied delay factor is loaded into the computer's Y register and is decremented until a value of zero is obtained. At that point, the counter is incremented. The counter actually consists of two memory locations, representing the number of counts of the timer in base 256. Location 988, the low-byte counter, holds the count modulo 256, while Location 989, the high-byte counter, contains the integer part of the count divided by 256. When the delay factor has been decremented to zero, the low-byte counter is incremented. When this location has been incremented 256 times from its initial value of zero, it overflows and the high-byte counter is incremented. The program is time balanced, so that the same number of machine cycles is taken regardless of whether the second incrementing is necessary.

If the second incrementing operation causes the high-byte counter to overflow, the overflow location is given a value of one. After each incrementing operation, Memory Location 49152 (C000 hex) is examined. A value greater than 127 in this location indicates that the keyboard has been actuated, in which case the routine clears the keyboard strobe and returns control to the calling program. Otherwise, a branch is taken to the beginning of the subroutine, and the counting continues if the overflow location contains a zero. If this location contains a one, the program continues to poll Location 49152 for keyboard activity, but no incrementing of the counters takes place.

The time interval measured by each "tick" of the routine depends upon two factors. One is the execution time of commands that will always be executed (check keyboard for activity, compare memory locations), which is 42 cycles of the microprocessor Phase 2 clock, each cycle being 9.7779E-07 sec in duration. The other factor is the time required to decrement the user-specified delay factor. For each unit change in the delay factor, five machine cycles are added to the execution time of the null loop. The minimum time interval that can be measured by this subroutine is therefore approximately 41 microsec, corresponding to a delay factor of one. (Use of a delay factor of zero is discouraged, because a decrementing operation will precede the test that determines whether the loop is finished. Thus the value in the accumulator will not be zero when the test is first performed.) At the other extreme, the maximum interval between updates of the counter corresponds to a delay factor of 255 and represents approximately 1.28 msec. In general, if DF is the value of the delay factor specified by the user and RF is the "resolution factor" representing time per update, then

$$RF = (5 * DF + 37)(9.7779E-07)$$

will yield a good approximation of the time per count.

This formula may be solved for DF in terms of RF in order to obtain a delay factor that will give a desired resolution. For example, millisecond resolution can be obtained by the use of $DF = 197$. Since the exact operating characteristics of the computer may vary from one installation to another (e.g., line voltage, component quality), it is wise to empirically determine the actual conversion relationship, by comparing the count returned from the subroutine with a timer of known accuracy, over a long time interval.

The subroutine's count is obtained by multiplying the value in the high-byte location by 256 and adding the value in the low-byte location, for example:

$$COUNT = 256 * PEEK(989) + PEEK(988).$$

If the counters have overflowed, a value of zero will result. The calling program can convert the count to seconds via the known resolution factor, although the count itself is sufficient for data analysis and less prone to propagated error. The maximum time that can be recorded is limited by the fact that the subroutine can increment the two counter locations 65,535 times before they overflow. After that point, the program will run, but the returned count will be incorrect. Thus the minimum delay factor of one allows timing of events up to 2.69 sec in duration, while the maximum delay of 255 allows times as long as 83.75 sec. Longer intervals may be obtained by the addition of a third counter location in a manner similar to Lines 3BE-3C5 in the listing. Of course, the additional execution time must be considered when resolution factors are estimated.

The timing routine that has been illustrated may be used for timing response latencies relative to actuating the computer's keyboard. As it stands, the timer can be used in two modes. In a stopwatch mode, the counter locations are reset to zero prior to each call to the subroutine. The value subsequently returned is the time for a single response. However, a net-time mode can be implemented by zeroing the counters prior to the first call only, in which case the routine returns the total time for all responses subsequently made. The BASIC statements that implement these two operations are illustrated in Figure 3.

In some applications, it may be desirable not only to measure response time, but also to record the actual response made. An example might be the answer chosen by a student to a multiple-choice or true-false question. In such cases, the instructions in 3D4-3D6 should be deleted so that the keyboard input can be recovered by a GET statement after the subroutine call in the BASIC program. Other sources of input may be used to interrupt the timer by changing the memory location probed in Lines 968-970.

Intertrial intervals, stimulus presentations, and keyboard lockouts are examples of situations in which a delay of a given length is needed. Instead of waiting for

```
(a)
10 POKE 986,0
20 POKE 987,DF
30 FOR I = 1 TO NSTIM
40 POKE 988,0
50 POKE 989,0
.
stimulus presentation

100 CALL 944
110 COUNT(I) = 256 * PEEK(989) + PEEK(988)
120 NEXT I

(b)
10 POKE 986,0
20 POKE 987,DF
30 POKE 988,0
40 POKE 989,0
50 FOR I = 1 TO NSTIM
.
stimulus presentation
.
100 CALL 944
110 NEXT I
120 COUNT = 256 * PEEK(989) + PEEK(988)
```

Figure 3. BASIC programs to implement the timing subroutine in (a) stopwatch and (b) net time modes.

keyboard activity to interrupt the timer, a subroutine can count until a user-specified limit is reached. Such a delay subroutine is shown in Figure 4, with a BASIC program for loading it in Figure 5. The delay timer program has been adjusted so that the previous formulas for delay factor and resolution factor apply here as well. In practice, the user "POKEs" the delay factor into Location 979 and the time limit, or number of executions of the routine, into Locations 980 and 981. Location 981 is loaded with the quotient of the time limit divided by 256, and Location 980 is loaded with the remainder. Figure 6 illustrates a BASIC program that will accomplish this task. Although the delay timer is

Hex(Dec)	Assembly	Machine	Comments
Address	Language	Language	
380 (944)	LDX #500	A2 00	clear X-register
387 (946)	LDY #500	A6 00	clear Y-register
384 (948)	SEC	18	enable subtraction
38F (949)	LDA \$03D3	AD D3 03	load delay factor
3E9 (952)	SBC #501	D9 01	decrement delay
38A (954)	BNE \$03AB	9C 0C	if non-zero, go back
3BC (956)	INX	1E	increment low byte
3BD (957)	BCD \$03AB	9C 0C	if zero, increment high byte
3BF (959)	JMP \$03A7	9C 0A	skip incrementing Y
3C2 (962)	INY	1F	increment high byte
3C3 (963)	CPX \$03D4	ED D4 03	compare X to low limit
3C6 (966)	BCD \$03DC	9C 0C	if equal, compare Y
3C8 (968)	NOP	EA	no operation
3C9 (969)	JMP \$03CF	9C 0F	skip Y comparison
3CC (972)	CPY \$03D5	ED D5 03	compare Y to high limit
3CF (975)	BNE \$03B4	9C 04	if unequal, start over
3D1 (977)	NOP	EA	no operation
3D2 (978)	RTS	6C	return to calling program
3D3 (979)			delay factor
3D4 (980)			low-byte limit
3D5 (981)			high-byte limit

Figure 4. Machine code and assembly language listings of the delay timer subroutine.

```

10 DATA 162,0,160,0,56,173,211,3,233,1,208,252
20 DATA 232,240,3,76,195,3,200,236,212,3,240,4
30 DATA 234,76,207,3,204,213,3,208,227,234,96
40 FOR I = 944 TO 973
50 READ X
60 POKE I,X
70 NEXT I
80 END

```

Figure 5. BASIC program to load the delay timer into memory.

shown using the same memory locations as the previous timer, both routines can be loaded at the same time by changing the location of one of them. Since both cannot be used at the same time, counter or DF locations can be shared to save space.

```

10 POKE 979,DF
20 INPUT "DELAY LIMIT";DL%
30 LH% = DL%/256
40 LL% = DL% - 256*LH%
50 POKE 980,LL%
60 POKE 981,LH%
70 CALL 944

```

other program statements

Figure 6. BASIC program to implement the delay timer.

CONCLUSIONS

Two very simple timing routines that can be used with a variety of microcomputers have been outlined here. Similar programs can be devised for other processors, and more involved routines can be built around the technique illustrated here. In some situations, a hardware clock may represent the best option. For a large set of experimental settings, however, a few minutes at the keyboard with, at most, 84 bytes of memory will provide the experimenter a great deal of flexibility in time-related manipulations.

REFERENCES

- FELSENSTEIN, L. The absolute time clock. *ROM*, 1978, 1(9), 58-65.
- GOLDBAND, S. Real-time clocks for microcomputers in behavioral research. *Behavior Research Methods & Instrumentation*, 1979, 11, 111-114.
- HOGENSON, J. Can your computer tell time? In D. H. Ahl & C. T. Helmers, Jr. (Eds.), *Best of Byte* (Vol. 1). Morristown, N.J.: Creative Computing Press, 1977.
- ROSNER, R. "Daddy, is it the PET?" *ROM*, 1978, 1(9), 26-32.

(Received for publication July 17, 1979;
revision accepted October 2, 1979.)