Behavior Research Methods & Instrumentation
1983, 15 (4), 459-464

Limitations of high-level microcomputer languages in software designed for psychological experimentation

JOSEPH G. DLHOPOLSKY
St. John's University, Staten Island, New York

The execution times of microcomputer high-level-language commands can be long enough to be of concern in experiments in which precise timing is a consideration. The problems in developing standard BASIC timing routines are addressed. A technique for using the Model III TRS-80 real-time clock to calibrate BASIC timinig routines is described, and representative execution times of selected commands are reported. It is concluded that high-level languages are too slow and that execution times are too variable for critical timing in experiments. On the other hand, machine language programs can provide the needed precision and control.

Timing with millisecond precision is a major requirement for microcomputers used in research. The timing of stimulus exposures, interstimulus intervals, and response latencies can be accomplished by a software timer or by having the computer input signals from an external clock. Each technique has advantages and disadvantages. An external clock is reliable, but it is also an additional expense and must be interfaced with the computer. A software timer requires no additional equipment, but it can be sensitive to tampering. Moreover, timers written in high-level languages (e.g., BASIC or PASCAL) are too slow for millisecond precision. This has led some researchers to rely on external timers for critical timing. Regardless of the timer used, however, reliable event timing also depends on the execution time of program commands, whether they are written in a high-level language or in machine language.

For purposes of this discussion, assume that an experiment consists of a series of discrete trials on which a stimulus is displayed, timed, and then masked. Also, a latency timer starts at stimulus onset and stops when the subject responds. Millisecond timing is required for stimulus exposure, interstimulus interval, and response latency: Either an external timer or a machine language software timer may be used for this.

A detailed analysis of trial events shows that a reliable timer is not all that is needed to assure that program timing fits the intended specifications. With a software timer, the following operations must be performed: (1) display the stimulus, (2) start timing response latency, (3) time stimulus exposure, (4) display the mask, (5) monitor the subject's response, and (6) stop and read the latency timer. With an external timer, the operations are these: (1) read the clock for the start time, (2) display the stimulus, (3) monitor the clock for the end of the exposure interval, (4) display the mask, (5) monitor the subject's response, and (6) read the elapsed time off the clock for response latency.

In both cases, the time it takes to display the stimulus must be accounted for in determining stimulus exposures and response latencies. This time will be short when controlled by machine language, but may be surprisingly long for a high-level language. Stimulus-display time also varies with the amount of material in the stimulus: A large stimulus consists of more data and takes longer to display than a short one. Furthermore, programs written in a high-level language cannot be synchronized with the video refresh rate, making it impossible to know when, within the 17-msec screen-scan time, the stimulus appears on the screen. The same problems affect displaying a stimulus mask and clearing the screen (which is done by displaying blanks). Other functions—reading a clock, monitoring for a keyboard or voice response, stopping and starting a clock—have execution times that must be considered when calibrating intervals in the experiment.

The execution times of machine language commands are tied to the microprocessor-chip architecture and are determined by the frequency of the microcomputer's system clock. This information is readily available (Zaks, 1980a, 1980b). Machine language command execution is fast, usually under 11 microsec. As long as the programmer accounts for interrupt processing, intervals programmed in machine language may be calibrated within microseconds of the desired interval. Furthermore, video synchronization can be accomplished on some computers by machine language techniques (Dlhopolsky, 1982; Grice, 1981; Merikle, Cheesman, & Bray, 1982; Reed, 1979). These techniques allow precise control over the appearance of stimulus displays and response timing because they synchronize the storing of stimulus information in video random access

The author's mailing address is: Department of Psychology, St. John's University, Staten Island, New York 10301.

memory (RAM) with the location of the screen's electron beam. Without video synchronization, response latency data will have a random error of up to 8.3 msec (Lincoln & Lang, 1980). High-level languages, on the other hand, do not share the precision of machine language: Execution times are not available, except for crude benchmark testing programs; execution is slower; and video synchronization is not possible.

The speed and precision of machine language make it clearly desirable for controlling experiment events. But, it must be recognized that many programs are written largely in BASIC or in another high-level language. The remainder of this article addresses the vagaries of BASIC that must be considered in designing software for experimentation. Although TRS-80 BASIC is addressed, the problems are not limited to that computer and that language. They are generic to all microcomputers and to all high-level languages.

PROBLEMS WITH BASIC STANDARDS

If the execution times of BASIC commands were known, critical program segments could be calibrated by summation, just as with T-states for machine language. Unfortunately, this information is not available, for a number of reasons. Foremost is that such information is unnecessary for word processing, balancing checkbooks, or saving the galaxy from invaders. Experimental psychologists have needs that are more esoteric.

Other than a general lack of interest in the execution times of BASIC commands, there are problems in developing timing standards. These are due to the plethora of different computers and the BASIC interpreters used. Several 8-bit microprocessors are in use today, the 6502 and Z-80 being the most popular. These microprocessors have different machine languages and operate at different speeds. This affects the timing of BASIC commands.

Even with computers that use the same microprocessor, the timing of BASIC commands cannot be expected to be the same. The microprocessor may not operate at the same speed in different models. For example, there are several variations of the Z-80 microprocessor, each with a different maximum clock speed. The Model I, Model III, and Model 4 TRS-80s use the Z-80 as the central processing unit (CPU). However, the system clock speed differs in all three. Even though the BASIC interpreters are likely to be similar, timing differs in each.

The proprietary nature of BASIC interpreters causes problems in arriving at standards. The secrecy surrounding the interpreters makes it likely that undocumented differences exist in the BASIC ROMs of computers produced by different manufacturers. This extends even to different models with the same microprocessor from the same manufacturer. Consider that the identical CLS (clear the video screen) instruction on the Model III takes about 36 msec to execute, whereas it takes 26 msec on the "slower" Model I. Lacking published standards for the execution times of high-level languages, the programmer has no choice but to calibrate intervals in each program before it is used for research.

CALIBRATING INTERVALS IN BASIC

In an attempt to determine the timing characteristics of a high-level language, the computer must be able to read real time before and after a to-be-calibrated program segment is executed. Not all microcomputers have internal real-time clocks that serve this purpose. The PET/CBM and the Model III TRS-80 have such clocks. The Model I TRS-80 has a real-time clock routine that is available with an expansion interface. In addition, some hardware modifications have been described for this model (Grice, 1981; Mapou, 1982). Microcomputers without real-time clocks—the Apple II, for example—may use an expansion port configured to read an external timer (Reed, 1979). A similar technique may be used with computers with built-in real-time clocks. The technique described in this article was developed for the Model III TRS-80, using its real-time clock.

The procedure for calibrating a BASIC segment is done by editing the to-be-calibrated segment so that the clock is read before and after the segment is executed. If the timing of the segment is very short, it may be too brief to register any elapsed time on the computer's real-time clock. Therefore, an external millisecond timer may be used in such cases (if so, the computer-timer interface should not use electromechanical relays). Alternatively, a software solution entails repeating the segment many times in a FOR/NEXT loop to acquire an average figure. The program listing in Figure 1 uses this technique to calibrate a .1-sec interval (timed in line 120).

This program works on the Model III TRS-80, with or without a disk-operating system. Variable A$ is set to the starting time by the TIME$ function. Line 120 is executed 10,000 times by the FOR/NEXT loop. Then variable B$ is set to the time at finish. The timing of one cycle through the 10,000 FOR/NEXT loop is determined from the difference between B$ and A$ divided by 10,000. Note that the resultant time includes the execution time of a single FOR/NEXT step, which must be subtracted to get the actual timing of line 120. In the Model III TRS-80, a single loop takes 1.39 msec

```
100 A$ = TIME$ : ' Reads clock
110 FOR J = 1 TO 10000 : ' Sets up 10000 repetitions
120 FOR J0 = 1 TO 71 : NEXT : ' Times .1 sec
130 NEXT
140 B$ = TIME$ : ' Reads time at end
```

Figure 1. A BASIC program that illustrates a routine for calibration of program segments. The segment to be calibrated is line 120.

for an integer looping variable and 2.19 msec for a single precision variable (found by using the same technique to measure an empty FOR/NEXT loop). Note that A$ and B$ are string variables and cannot be directly subtracted. The timer in line 120 is for demonstration only. As written, it will not accurately time .1 sec. Each of the spaces, added here for readability, will add .02 msec to the time it takes to complete one loop (in the Model III TRS-80). The looping constant (71) in line 120 can be changed until the user is satisfied that the interval is being timed with reasonable accuracy. Once this is done, the calibration lines (lines 100, 110, 130, and 140) can be deleted before the program is used for research.

Figure 2 contains the listing of a program, BASIC TIMING CALIBRATION, that uses the described

```
10 REM  BASIC TIMING CALIBRATION
        By Joseph G. Dlhopolsky, Ph.D.
        St. John's University
        Staten Island, New York 10301

20 REM  Revised 8301.17

30 GOTO100
40 RETURN
90 CLEAR1000
95 DEFSTRA-H:DEFINTI-N:DEFDBLX-Z:REM  Sets variable types
96 CMD"L","BASTIME/CMD":REM  Loads machine language routines
97 DEFUSR0=&HFFD0:REM  Clock reset routine location
98 DEFUSR1=&HFFE3:REM  Clock read routine location
100 I=10000:REM  Sets number of FOR:NEXT loops
101 A0="X":A1="1234567890":A2=STRING$(64,45):REM  Sets string
    variables
102 CLS
103 PRINT"Do you want hard copy?";
104 GOSUB9400
105 CLS:GOSUB910
106 POKE&HFFCF,IC
107 PRINT"DURATION OF BASIC INSTRUCTIONS (commands on one
    program line)":PRINT
108 J=USR0(0):FORO=1TOI:NEXT:J=USR1(0)
109 PRINT"Empty single precision loop:";:GOSUB810
110 J=USR0(0):FORJ=1TOI:NEXT:J=USR1(0)
111 PRINT"Empty integer loop:";:GOSUB810
112 PA=J2%6+J1/10+J0/300:REM  Correction for FOR:NEXT loop
    overhead
115 PRINT
116 PRINT"Remaining values are corrected for duration of 10000
    cycle integer FOR/NEXT loop.":PRINT
120 J=USR0(0):FORJ=1TOI:      NEXT:J=USR1(0)
122 PRINT"Five additional spaces in command:";:GOSUB810
125 J=USR0(0):FORJ=1TOI:CLS:NEXT:J=USR1(0)
130 PRINT"CLS:";:GOSUB810
140 J=USR0(0):FORJ=1TOI::NEXT:J=USR1(0)
150 PRINT"Additional colon:";:GOSUB810
170 DEFUSR3=&HFFE2:REM  Sets up RET loop
180 J=USR0(0):FORJ=1TOI:J0=USR3(0):NEXT:J=USR1(0)
185 PRINT"USR3(0) call to RET:";:GOSUB810
190 J=USR0(0):FORJ=1TOI:REM
200 NEXT:J=USR1(0)
205 PRINT"REM at line end:";:GOSUB810
210 J=USR0(0):FORJ=1TOI:REM1234567890
220 NEXT:J=USR1(0)
221 PRINT"REM and 10 characters on line end:";:GOSUB810
222 J=USR0(0):FORJ=1TOI
    :REM12345678901234567890123456789012345678901234567890
224 NEXT:J=USR1(0)
226 PRINT"REM and 50 characters on line end:";:GOSUB810
228 GOSUB900
230 J=USR0(0):FORJ=1TOI:PRINTJ;:NEXT:J=USR1(0):GOSUB910
231 CLS:PRINT"PRINTJ; (integer):";:GOSUB810
232 GOSUB900
233 J=USR0(0):FORJ=1TOI:PRINTA0;:NEXT:J=USR1(0):GOSUB910
234 CLS:PRINT"PRINTA0; (one character string):";:GOSUB810
235 GOSUB900
236 J=USR(0):FORJ=1TOI:PRINTA1;:NEXT:J=USR1(0):GOSUB910
237 CLS:PRINT"PRINTA1; (ten character string):";:GOSUB810
238 GOSUB900
239 J=USR0(0):FORJ=1TOI:PRINTA2;:NEXT:J=USR1(0):GOSUB910
240 CLS:PRINT"PRINTA2; (64 character string):";:GOSUB810
241 GOSUB900
242 J=USR0(0):FORJ=1TOI:PRINTJ:NEXT:J=USR1(0):GOSUB910
243 CLS:PRINT"PRINTJ (integer):";:GOSUB810
244 GOSUB900
245 J=USR0(0):FORJ=1TOI:PRINTA0:NEXT:J=USR1(0):GOSUB910
246 CLS:PRINT"PRINTA0 (one character string):";:GOSUB810
247 GOSUB900
248 J=USR0(0):FORJ=1TOI:PRINTA1:NEXT:J=USR1(0):GOSUB910
249 CLS:PRINT"PRINTA1 (ten character string):";:GOSUB810
250 GOSUB900
251 J=USR0(0):FORJ=1TOI:PRINTA2:NEXT:J=USR1(0):GOSUB910
252 CLS:PRINT"PRINTA2 (64 character string):";:GOSUB810
253 GOSUB900
254 J=USR0(0):FORJ=1TOI:PRINT@0,J;:NEXT:J=USR1(0):GOSUB910
255 CLS:PRINT"PRINT@0,J; (integer):";:GOSUB810
256 GOSUB900
257 J=USR0(0):FORJ=1TOI:PRINT@0,A0;:NEXT:J=USR1(0):GOSUB910
258 CLS:PRINT"PRINT@0,A0; (one character string):";:GOSUB810
259 GOSUB900
260 J=USR0(0):FORJ=1TOI:PRINT@0,A1;:NEXT:J=USR1(0):GOSUB910
261 CLS:PRINT"PRINT@0,A1; (ten character string):";:GOSUB810
262 GOSUB900
263 J=USR0(0):FORJ=1TOI:PRINT@0,A2;:NEXT:J=USR1(0):GOSUB910
264 CLS:PRINT"PRINT@0,A2; (64 character string):";:GOSUB810
265 GOSUB900
266 J=USR0(0):FORJ=1TOI:PRINT@0,J:NEXT:J=USR1(0):GOSUB910
267 CLS:PRINT"PRINT@0,J (integer):";:GOSUB810
268 GOSUB900
269 J=USR0(0):FORJ=1TOI:PRINT@0,A0:NEXT:J=USR1(0):GOSUB910
270 CLS:PRINT"PRINT@0,A0 (one character string):";:GOSUB810
271 GOSUB900
272 J=USR0(0):FORJ=1TOI:PRINT@0,A1:NEXT:J=USR1(0):GOSUB910
273 CLS:PRINT"PRINT@0,A1 (10 character string):";:GOSUB810
274 GOSUB900
275 J=USR0(0):FORJ=1TOI:PRINT@0,A2:NEXT:J=USR1(0):GOSUB910
276 CLS:PRINT"PRINT@0,A2 (64 character string):";:GOSUB810
277 GOSUB900:JA=1
278 J=USR0(0):FORJ=1TOI:PRINT@JA,J;:NEXT:J=USR1(0):GOSUB910
279 CLS:PRINT"PRINT@JA,J; (both integers):";:GOSUB810
280 GOSUB900
281 J=USR0(0):FORJ=1TOI:PRINT@JA,A0;:NEXT:J=USR1(0):GOSUB910
282 CLS:PRINT"PRINT@JA,A0; (integer address, one character
    string:";:GOSUB810
283 GOSUB900
284 J=USR0(0):FORJ=1TOI:PRINT@JA,A1;:NEXT:J=USR1(0):GOSUB910
285 CLS:PRINT"PRINT@JA,A1; (integer address, ten character
    string:";:GOSUB810
286 GOSUB900
287 J=USR0(0):FORJ=1TOI:PRINT@JA,A2;:NEXT:J=USR1(0):GOSUB910
288 CLS:PRINT"PRINT@JA,A2; (integer address, 64 character
    string):";:GOSUB810
289 J=USR0(0):FORJ=1TOI:POKE15360,0:NEXT:J=USR1(0)
290 PRINT"One POKE15360,0:";:GOSUB810
291 JA=0
292 J=USR0(0):FORJ=1TOI:POKE15360,JA:NEXT:J=USR1(0)
293 PRINT"One POKE15360,JA:";:GOSUB810
294 J=USR0(0):FORJ=1TOI:POKE&H3C00,0:NEXT:J=USR1(0)
295 PRINT"POKE&H3C00,0:";:GOSUB810
296 J=USR0(0):FORJ=1TOI:POKE&H3C00,JA:NEXT:J=USR1(0)
297 PRINT"POKE&H3C00,JA";:GOSUB810
298 JA=15360:JB=0
299 J=USR0(0):FORJ=1TOI:POKEJA,0:NEXT:J=USR1(0)
300 PRINT"One POKEJA,0:";:GOSUB810
301 J=USR0(0):FORJ=1TOI:POKEJA,JB:NEXT:J=USR1(0)
302 PRINT"One POKEJA,JB:";:GOSUB810
303 J=USR0(0):FORJ=1TOI:A=INKEY$:NEXT:J=USR1(0)
304 PRINT"A=INKEY$:";:GOSUB810
305 OUT236,48
306 J=USR0(0):FORJ=1TOI:OUT0,0:NEXT:J=USR1(0)
307 PRINT"OUT0,0:";:GOSUB810
308 J=USR0(0):FORJ=1TOI:J0=INP(0):NEXT:J=USR1(0)
309 PRINT"J0=INP(0):";:GOSUB810
310 J=USR0(0):FORJ=1TOI:J0=PEEK(15360):NEXT:J=USR1(0)
311 PRINT"J0=PEEK(15360):";:GOSUB810
312 J=USR0(0):FORJ=1TOI:J0=PEEK(&HFFFF):NEXT:J=USR1(0)
313 PRINT"J0=PEEK(&HFFFF):";:GOSUB810
314 J1=15360
315 J=USR0(0):FORJ=1TOI:J0=PEEK(J1):NEXT:J=USR1(0)
316 PRINT"J0=PEEK(J1):";:GOSUB810
317 J=USR0(0):FORJ=1TOI:GOSUB40:NEXT:J=USR1(0)
318 PRINT"GOSUB40 (40 RETURN):";:GOSUB810
319 J=USR0(0):FORJ=1TOI:GOSUB9420:NEXT:J=USR1(0)
320 PRINT"GOSUB9420 (10010 RETURN):";:GOSUB810
350 DIMX(3400):JC=15360:JD=0
352 J=USR0(0):FORJ=1TOI:POKEJC,JD:NEXT:J=USR1(0)
355 PRINT"One POKEJC,JD on top of 3400 item double precision
    array:";:GOSUB810
799 CLS:CMD"Z","OFF":END
810 J0=PEEK(&HFFFD):REM  Time base
815 J1=PEEK(&HFFFE):REM  Seconds
820 J2=PEEK(&HFFFF):REM  Minutes
825 J3=30-J0:PRINTJ2%6+J1/10+J3/300-PA;"msec":GOTO9200
900 CMD"Z","OFF":RETURN
910 IFIC=1THENCMD"Z","ON"
920 RETURN
9100 A=INKEY$
9110 A=INKEY$:IFA=""THEN9110ELSERETURN
9200 IFIC=1THENGOTO910ELSEGOSUB900:A=INKEY$
9210 PRINT"  ( P R E S S    A N Y    K E Y    T O    C O N T I
    N U E )";
9220 FORI=1TO300:A=INKEY$
9230 IFA=""THENNEXT:PRINTCHR$(29);CHR$(30);CHR$(29);:GOTO9210
9240 PRINTCHR$(29);CHR$(30);CHR$(29);:GOTO910
9400 PRINT"  (Y)es or (N)o?  ";
9410 GOSUB9100:IFA="Y"THENIC=1:PRINT"Yes.";
    ELSEIFA="N"THENIC=2:PRINT"No.";ELSE9410
9420 RETURN
```

Figure 2. Source code for BASIC TIMING CALIBRATION, a BASIC program that times and reports the execution of selected TRS-80 BASIC commands.

technique for timing the duration of selected BASIC commands on the Model III TRS-80 under the TRSDOS 1.3 operating system. Because the program uses machine language subroutines, high memory must be protected in the normal fashion. When executed, the program displays or prints out the execution times for each command.[1]

In an attempt to arrive at values that are as precise as possible, the TRS-80's TIME$ function is not used in the program. Rather, two machine language routines are used: one for resetting the real-time clock and the other for reading the clock. The J=USR0(0) command calls the clock-reset routine, which sets the clock to zero, and J-USR1(0) reads the clock. The source code for the machine language routines is in Figure 3. It may be assembled on disk with the file specification of BASTIME/CMD. Alternatively, the hexadecimal object code in Column 2 of Figure 3 can be POKEd, as part of a BASIC routine, into the memory addresses that appear as hexadecimal codes in Column 1.

The clock-read routine is executed in 45.4 microsec and stops the clock while it transfers the real-time clock's time base, seconds, and minutes to nonvolatile memory locations. In the Model III TRS-80, the time base is a 33.3-msec clock that is maintained at address 4216 hexadecimal (hex) or 16918 decimal (dec). Seconds and minutes are stored at addresses 4217 hex (16919 dec) and 4218 hex (16920 dec), respectively.

The clock-read routine was written for a 48K Model III TRS-80. The hexadecimal values in lines 270, 290, and 310 may be changed to lower addresses for Model III computers that have smaller RAMs. This is done by changing the most significant byte (MSB), which is FF, to BF for computers with 32K of RAM or 7F for computers with 16K. If the TRSDOS operating system is unavailable, line 96 of BASIC TIMING CALIBRATION may be deleted and the A=TIME$ function may replace all the J=USR0(0) and J=USR1(0) statements. The results should be reasonably close.

```
                00100 ;CLOCK READ AND RESET ROUTINE FOR BASTIME/BAS
                00110 ;By Joseph G. Dlhopolsky, Ph.D.
                00120 ;Revised 8301.17
                00130
FFD0            00140         ORG     0FFD0H
                00150 ;USR0: CLOCK RESET
FFD0 F3         00160         DI                      ;Stops clock
FFD1 3E00       00170         LD      A,0
FFD3 321742     00180         LD      (16919),A       ;Reset seconds
FFD6 321842     00190         LD      (16920),A       ;Reset minutes
FFD9 3E1E       00200         LD      A,30
FFDB 321642     00210         LD      (16918),A       ;Reset time base
FFDE FB         00220         EI                      ;Starts clock
FFDF C9         00230         RET                     ;Return to BASIC
                00240 ;USR1: READ CLOCK & STORE IN HIGH MEMORY
FFE0 F3         00250         DI                      ;Stops clock
FFE1 3A1642     00260         LD      A,(16918)       ;Get 1/30 second
FFE4 32FDFF     00270         LD      (0FFFDH),A      ;Store 1/30 second
FFE7 3A1742     00280         LD      A,(16919)       ;Get seconds
FFEA 32FEFF     00290         LD      (0FFFEH),A      ;Store seconds
FFED 3A1842     00300         LD      A,(16920)       ;Get minutes
FFF0 32FFFF     00310         LD      (0FFFFH),A      ;Store minutes
FFF3 FB         00320         EI                      ;Starts clock
FFF4 C9         00330         RET                     ;Return to BASIC
FFD0            00340         END     0FFD0H
00000 Total Errors
```

Figure 3. Machine language source code for the clock-reset and clock-read routines used by BASIC TIMING CALIBRATION. The hexadecimal values for the object code appear in the second column. The proper memory addresses for the object code are listed in the first column.

For use on the Model I TRS-80, the clock addresses in the machine language routines must be changed. In the real-time clock provided with Radio Shack's expansion interface, the time base is a 25-msec clock that is maintained at address 405E hex (16478 dec), with seconds and minutes at 405F hex (16479 dec) and 4060 hex (16480 dec), respectively. The BASIC TIMING CALIBRATION program must also be rewritten to make it compatible with the Model I's real-time clock and disk-operating system. Of special note is line 825, which takes the elapsed time, divides by 10,000, and subtracts the time taken by the FOR/NEXT loop (variable PA). Since the time base is maintained differently in the two models, line 825 must be changed for the Model I. In the Model III TRS-80, the time base counts down from 30 to 1. So the elapsed time in the time base is equal to the product of 33.3 msec and the difference of 30 and the current reading. In the Model I TRS-80, the time base counts up from 0 to 39. Thus, the elapsed time is the current reading multiplied by 25 msec.

For programmers who would like to use the demonstration program on computers other than TRS-80 models, note line 95. This line defines ranges of variables as integers, strings, and double-precision. Specifically, all variable names beginning with the letters A through H are arbitrarily defined as string variables, those from I through N as integers, and those from X through Z as double-precision. These conventions save programming time and computer memory, but they are not universal. For example, the Apple II does not allow definition of variable ranges and does not support double-precision variables. The standard symbols for strings ($) and integers (%) must be appended to the appropriate variable names for the program to transfer to the Apple II. The means by which the real-time clock is read and the elapsed time calculated must also be modified.

RESULTS OF CALIBRATION ATTEMPTS

Table 1 lists the values produced by the BASIC TIMING CALIBRATION program on the author's Model III TRS-80. These values should be considered advisory, not standard. As can be seen from Table 1, there is considerable variability in execution times.

Surprisingly, it takes 35.7 msec to execute a CLS (clear the screen) instruction. This is slightly longer than two full scans of the screen's electron beam (16.7 msec each). In order to clear the screen, the CPU loads blanks in all 1,024 video RAM locations. However, the CLS instruction also carries out other functions, since a machine language program can be written to clear the screen in 10.6 msec (Dlhopolsky, 1982). In any event, it is not certain when in the 35.7-msec interval the screen begins to be cleared and when it is completed. This makes the CLS instruction a poor choice for erasing stimuli. If machine language cannot be used, it would be better to print over the to-be-erased stimulus

Table 1
Representative Data From the BASIC TIMING CALIBRATION Program

BASIC FUNCTION	ESTIMATED TIME	
CLS	35.7 msec	
USRØ(Ø)	4.Ø msec	(a)
REM (alone)	1.8 msec	
REM + 10 characters	2.1 msec	
REM + 50 characters	3.4 msec	
POKE15360,Ø	3.9 msec	
POKE15360,JA	5.1 msec	
POKEJA,Ø	4.1 msec	
POKEJA,JB	5.4 msec	
POKEJC,JD	5.8 msec	
POKE&H3CØØ,Ø	2.9 msec	(a)
POKE&H3CØØ,JA	4.1 msec	(a)
JØ=PEEK(15360)	5.1 msec	
JØ=PEEK(J1)	5.Ø msec	
JØ=PEEK(&HFFFF)	4.1 msec	(a)
OUTØ,Ø	2.9 msec	
JØ=INP(Ø)	4.2 msec	
A=INKEY$	3.7 msec	
GOSUB (to low line)	4.4 msec	
GOSUB (to high line)	18.7 msec	
PRINTJ	26.6 msec	
PRINTJ;	9.8 msec	
PRINTØØ,J	13.2 msec	
PRINTØØ,J;	9.2 msec	
PRINTØJA,J;	1Ø.4 msec	
PRINTAØ	21.4 msec	
PRINTA1	26.1 msec	
PRINTA2	71.5 msec	
PRINTAØ;	3.2 msec	
PRINTA1;	1Ø.3 msec	
PRINTA2;	52.8 msec	
PRINTØØ,AØ	8.Ø msec	
PRINTØØ,A1	12.6 msec	
PRINTØØ,A2	39.9 msec	
PRINTØØ,AØ;	4.Ø msec	
PRINTØØ,A1;	8.6 msec	
PRINTØØ,A2;	35.9 msec	
PRINTØJA,AØ;	5.2 msec	
PRINTØJA,A1;	9.9 msec	
PRINTØJA,A2;	37.1 msec	
Five additional spaces	.1 msec	
One unneeded colon	.8 msec	

Note–J variables are integers. A0, A1, and A2 are string variables of 1, 10, and 64 characters, respectively. Commands marked (a) are TRSDOS enhancements to BASIC.

with a string variable consisting of blank characters. This, of course, assumes that the stimulus does not occupy the entire screen.

The USR0(0) command calls a machine language subroutine under TRSDOS. To test only the execution time of the USR0(0) command, the first command in the called routine is 201, which executes an immediate return to BASIC. The call and return from the machine language routine entails 4.0 msec, which must be considered whenever a machine language routine (e.g., a millisecond timer) is called from a parent BASIC program.

The GOSUB command can have variable execution times depending on where in the program the intended subroutine is located. The computer performs a serial search through the object code for the line number of the called subroutine. Subroutines early in the program are found more quickly than those situated later. There is a further difficulty in the GOSUB command: The location of the line in which the GOSUB is found also seems to have an effect on its execution time.

Remark statements are generally considered to be "transparent" in that they carry out no program function. However, the REM (indicated as ') statement alone has been found to add 1.8 msec to program execution time. The length of the remark message also extends execution time. A regression analysis of the number of characters in the three REM statements of BASIC TIMING CALIBRATION indicates that each character adds an additional .032 msec to the execution time (Table 2). This is in addition to using up 1 byte of memory per character.

Other incidentals include extra spaces and unneeded colons that sometimes find their way into programs during editing. In general, each character uses 1 byte of memory. Beyond the extra memory used, however, it takes time for the CPU to skip over the extraneous material. Each space adds .02 msec to execution time and colons add .8 msec.

In cases in which the researcher decides against machine language control of stimulus displays, a variation of the PRINT command often is used. Even when machine language control is used for this purpose, a ready signal or fixation point might be displayed by BASIC during an intertrial interval. The PRINT command has more variations than any other BASIC command, and the variations have idiosyncratic execution times.

In most cases, stimuli can be displayed as strings, less frequently as integers, and rarely as single-precision. When PRINTing any kind of variable, the computer must perform a serial search through memory for the name of the variable and its location in memory. To reduce this time, it is best to define numerical variables with dummy statements early in the program. This gives them a preferred location in the variable list and subsequently cuts down on the execution time of a PRINT command. This technique, however, is not guaranteed to work with string variables. If a specific string variable is redefined during the course of a program, it is unlikely that it will be stored in the same memory location.

The time taken to display a fixation point or ready

Table 2
Results of Regression Analysis of Length of Remark Statements and Size of String Variables in PRINT Statements

COMMAND	CHARACTERS	CORRELATION	SLOPE	Y INTERCEPT
REM	Ø, 1Ø, 5Ø	+1.Ø	.Ø32	1.79
PRINTx	1, 1Ø, 64	+ .999	.811	19.39
PRINTx;	1, 1Ø, 64	+1.Ø	.788	2.46
PRINTØØ,x	1, 1Ø, 64	+1.Ø	.5Ø6	7.51
PRINTØØ,x;	1, 1Ø, 64	+1.Ø	.5Ø6	3.51
PRINTØJA,x;	1, 1Ø, 64	+1.Ø	.3Ø5	4.76

signal may be shortened by selecting the appropriate version of the PRINT command. As seen in Table 1, the longest execution time is claimed by the simple PRINT statement. To execute this statement, the computer looks up the cursor location, prints each character of the variable, clears the rest of the line, executes a line feed and carriage return, and clears the next line—26.6 msec for an integer and 21.4 msec for a single-character string. Except for actually printing the stimulus, the rest of these functions are usually unnecessary. A simple change in the format of the command eliminates the extraneous processes. The PRINT@0,J; command prints an integer in 9.2 msec, whereas the PRINT@0,A0; command prints a single-character string in 4.0 msec. In both cases, the command provides the cursor location, and the semicolon suppresses the line feed and erasure of the two lines.

The display of a string variable presents a further problem in that each additional character in the string adds to the execution time. Table 2 shows the results of a regression analysis of the different PRINT commands with strings of character length 1, 10, and 64, (64 is the character length of one video line in the Models I and III TRS-80). Almost perfect correlations indicate that the execution time for printing a string can be accurately predicted from the string's length. An interesting note is that the same rule applies to printing a string variable made up of blanks, even though nothing appears on the screen.

DISCUSSION

It is apparent that the execution times of TRS-80 BASIC commands are difficult to establish. When one considers that the values described here are from one of the several models of the TRS-80 and that the TRS-80 is one of many computers available to researchers, the task of establishing standards for high-level languages is monumental.

A further concern is the ease with which the execution time of a program segment can be altered by seemingly innocuous factors: a change in command format, an intrusion of extraneous characters, the presence of remarks, the type of variable being used (string, integer, or floating point). In contrast, assembly language programming requires a restricted format, extraneous characters are impossible, remarks are truly transparent, and variable types cannot be assumed. The greater difficulty of programming in assembly language is a virtue in this sense.

Speed of operation is another concern. The fastest BASIC command was OUT0,0, at 2.9 msec, whereas others ranged close to 40 msec. Of particular concern is the dependency of the PRINT command on stimulus size. A one-line stimulus (64 characters) took 39.9 msec to complete. A stimulus twice as large would take close to 80 msec. In this case, the electron beam would have refreshed the screen almost five times while the stimulus was being drawn! Determining exactly when the stimulus is "officially" displayed is a problem, and the theoretical validity of such a determination is questionable. This makes BASIC inadvisable for experiments that require rapid stimulus displays and precise response latencies. By machine language standards, 3-40 msec is a snail's pace: The slowest Z-80 command is executed in 10.5 microsec on the Model III TRS-80. An added benefit of machine language is the ability to establish synchronization between the Z-80 execution of a stimulus display and the location of the video screen's electron beam. Thus, the machine language potential for precision and speed of execution allows a broader range of experiments to be programmed on the microcomputer, and the resultant data can be expected to have a firmer foundation.

A good research program need not be written entirely in assembly language. Most experiments can be divided into two functional parts: (1) a series of trials in which stimuli are displayed and responses are recorded, and (2) housekeeping functions (setting up the next trial, making permanent records of response data, display of messages to the subject, etc.). High-level language is suitable for housekeeping in cases in which variations in execution times are not expected to affect the dependent variable. Machine language should be used whenever such an effect is possible. Therefore, the standard research program should be a hybrid program that is written largely in a high-level language, with critical trial events being controlled by machine language.

REFERENCES

DLHOPOLSKY, J. G. Software synchronizing of video diplays and Z-80 processing in the Model III TRS-80. *Behavior Research Methods & Instrumentation*, 1982, 14, 539-544.

GRICE, G. R. Accurate reaction time research with the TRS-80 microcomputer. *Behavior Research Methods & Instrumentation*, 1981, 13, 674-676.

LINCOLN, C. E., & LANE, D. M. Reaction time measurement errors resulting from the use of CRT displays. *Behavior Research Methods & Instrumentation*, 1980, 12, 27-39.

MAPOU, R. L. Tachistoscopic timing on the TRS-80. *Behavior Research Methods & Instrumentation*, 1982, 14, 534-538.

MERIKLE, P. M., CHEESMAN, J., & BRAY, J. PET Flasher: A machine language subroutine for timing visual displays and response latencies. *Behavior Research Methods & Instrumentation*, 1982, 14, 26-28.

REED, A. V. Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, 1979, 11, 572-575.

ZAKS, R. *How to program the Z-80*. Berkeley, Calif: Sybex, 1980.

ZAKS, R. *Programming the 6502* (3rd ed.). Berkeley, Calif: Sybex, 1980.

NOTE

1. The author will provide a double-density floppy disk, formatted for TRSDOS 1.3 (but not containing TRSDOS itself), with BASIC TIMING CALIBRATION and the source and object codes for the associated machine language routines. The cost is $10. If you want a disk with the TRSDOS operating system, please so state and send verification that you already own TRSDOS.

(Manuscript received January 27, 1983;
revision accepted for publication August 12, 1983.)