COMPUTER TECHNOLOGY

A multipurpose laboratory package for OS/8 BASIC

JAMES H. HOWARD, JR.

Catholic University of America, Washington, D.C. 20064

A set of BASIC-callable assembler routines are described that support experimental programming on the PDP-8/e laboratory computer. The package includes both general-purpose laboratory utilities (e.g., for timing delays, sensing and controlling switches, etc.) and specialized functions to facilitate elementary signal processing operations. System performance is evaluated in terms of timing accuracy, optimal sampling rates, and transportability to other laboratories.

A versatile and easy-to-use software package is one of the single most important components of a computerized psychological laboratory. The present paper describes a set of BASIC-callable assembler routines designed to facilitate experimental programming in a PDP-8/e-based laboratory. The package was initially coded in 1975, and it has been used regularly since that time in a variety of cognitive and psychophysical experiments. Although the routines are obviously geared to our particular applications and system configuration, the package requires only a minimal OS/8 system, and individual subroutines can be modified easily to suit any unique hardware requirements.

A number of criteria were considered in designing the package to guarantee flexibility. (1) It was essential that the package handle the elementary signal processing requirements necessary for psychoacoustic research. These include sampling at data rates of at least 12.5 kHz, with sufficient processing power and precision to handle routine signal synthesis and analysis algorithms off-line. (2) It was important that the system have enough string processing capability to handle verbal material for reaction time (RT) and logical problem solving studies. (3) It was also necessary that we be able to time experimental events (both stimulus presentation and RTs) with as close to 1-msec accuracy as possible. (4) Finally, it was particularly important that the system be easy to use. We wanted to code and debug experimental control programs in minimal time (4-8 h for a well designed experiment), and to make the system accessible to

relatively unsophisticated users (undergraduate students with as little as 2-4 weeks of programming experience).

After examining a number of alternatives, including subroutine libraries for FORTRAN II or FORTRAN IV and a customized assembler-coded package, we decided to implement the system by coding our own multi-function user overlay for OS/8 BASIC. This seemed to be the most reasonable approach for a variety of reasons. First, the string handling and numeric processing capabilities were adequate for our purposes.¹ Although FORTRAN obviously has the upper hand in number-crunching, string functions are virtually nonexistent. In addition, the ability to code off-line support routines (e.g., acoustic stimulus generation) and control programs in the same language ensures stimulus and data file compatibility and generally facilitates programming.² Second, BASIC is simple to learn and use, and it has been implemented on most machines from main frames to micros. The latter feature provides students immediate access to other computers on this campus (DECSys 10, PDP-11s, PETs, etc.) and elsewhere. Third, OS/8 BASIC is set up to accept a user-written overlay without extraordinary difficulty. Although a set of LAB-8/e functions are provided with OS/8 BASIC, we found them to be inconsistent with our hardware configuration and processing requirements.

FUNCTION DEFINITIONS

Fifteen separate functions were coded that enabled BASIC programs to access our laboratory peripherals. The relevant laboratory peripherals are listed in Table 1.³ The fifteen routines are organized as two separate sets: general laboratory utilities and signal processing utilities. The mnemonic names and proper calling syntax for each function are indicated in Table 2.

General Laboratory Utilities

The first eight routines enable the user to perform

The author gratefully acknowledges the helpful comments of James Ballas, Donald Burgy, and Darlene Howard on an earlier version of this manuscript and thanks Richard Millward and Steven Myerow for helpful information provided on the OS/8 BASIC run-time system. Requests for reprints should be addressed to the author, Department of Psychology, Catholic University of America, Washington, D.C. 20064. Source listings of the software described here may be obtained for a nominal reproduction charge by writing the author.

Table 1
Supported PDP-8/e Laboratory Peripherals

DK8-EC	5-kHz Crystal Clock
AD-01	10-bit A/D Converter
AA-50	12-bit D/A Converter
DR8-EA	Digital Input/Output
KR8-E	Asynchronous Serial Interface

Note—Minor program changes would be required to substitute standard peripherals.

standard laboratory tasks such as timing a delay, setting or reading switches, displaying information on a subject's CRT, or reading from his or her terminal keyboard. These eight routines alone are sufficient to program a wide range of cognitive tasks. Each routine is described in detail below. As in Table 2, the variable X designates a dummy argument that is neither used nor changed by the user function.

(1) DEL—This function is used to impose a delay for timing stimulus presentations or interstimulus intervals. When the statement $X = \text{DEL}(N)$ is executed, the user package retains control for N msec ($0 < N < 4,096$). Timing is accomplished by counting ticks of the 5-kHz clock, but for convenience, the argument is passed in milliseconds.

(2) DRS—This function enables the user to set one or more output bits on the DR8-EA. The statement $X = \text{DRS}(N)$ outputs a bit pattern corresponding to the single-precision binary representation of the decimal argument N ($N < 4,096$). For example, to set output Bit 10, the function is called with $N = 2^{**}10 = 1,024$, whereas $N = 2^{**}10 + 2^{**}0 = 1,025$ is required to set Bits 10 and 0. Output bits set by previous DRS calls cannot be cleared with this function.

(3) DRC—This function completely parallels DRS, but clears rather than sets output bits in the DR8. As with DRS, single or multiple bits may be addressed. The first three functions, DEL, DRS, and DRC may be used to control events involving simple switch closures.

For example, the following sequence sets output Bit 10, waits 50 msec, sets output Bit 11, then clears both bits after an additional delay of 150 msec.

```
100 X = DRS(1024)
105 X = DEL(50)
110 X = DRS(2048)
115 X = DRS(150)
120 X = DRC(3072)
```

Only Bits 10 and 11 are influenced by this sequence.

(4) CRO—The statement $X = \text{CRO}(S\$)$ causes the string $S\$$ to be displayed on the subject's terminal. We use a standard Applied Digital Data Systems Consul 580 CRT at 9,600 baud. The function transmits OS/8 BASIC 6-bit format at a rate specified by the baud rate setting on the asynchronous interface. OS/8 BASIC storage conventions limit character strings to 80 or fewer uppercase characters; consequently, extended text such as logical word problems must be presented by multiple-function calls.

(5) CRC—This function parallels the CRO function, but it enables the user to output ASCII control characters (i.e., Bit 7 suppressed) to the subject's display. Since most CRT displays handle cursor control and other special functions (e.g., screen erase and line select) by means of these characters, and OS/8 BASIC uses an internal 6-bit character code, a separate control function was necessary. For example, on our terminal, a screen erase ("control L") is performed by $X = \text{CRC}("L")$. To illustrate the CRO and CRC functions, the following sequence displays "PART 1" on the CRT, waits 500 msec, then displays "PART 2" on the same line (no cursor movements are executed between successive CRO commands), waits 2 sec, and erases the screen.

```
100 X = CRO("PART 1")
105 X = DEL(500)
110 X = CRO(" PART 2")
115 X = DEL(2000)
120 X = CRC("L")
```

Table 2
Mnemonic, Calling Syntax, and Entry Points for Each Laboratory Routine

	Function	Calling Syntax	Entry Point	Description
1.	DEL	$X = \text{DEL}(N)$	3400	Delay N msec
2.	DRS	$X = \text{DRS}(N)$	3445	Set output bit(s)
3.	DRC	$X = \text{DRC}(N)$	3451	Clear output bit(s)
4.	CRO	$X = \text{CRO}(S\$)$	3455	Output string on CRT
5.	CRC	$X = \text{CRC}(S\$)$	3522	Control function on CRT
6.	DRI	$R = \text{DRI}(N)$	3531	Read input bit(s)
7.	GRT	$T = \text{GRT}(X)$	3741	Get reaction time for DRI
8.	CRI	$T = \text{CRI}(F\$,N)$	3600	Read string from CRT
9.	FIX	$X = \text{FIX}(M,N)$	4000	Convert from floating point to fixed point
10.	FLO	$M = \text{FLO}(N)$	4031	Convert from fixed point to floating point
11.	ADC	$X = \text{ADC}(M,N,C)$	4200	Sample A/D converter
12.	DAC	$X = \text{DAC}(M,N,T)$	4264	Output on D/A converter
13.	OFF	$X = \text{OFF}(I)$	4400	Offset initial index for fixed-point array
14.	ADD	$X = \text{ADD}(N,M)$	4464	Add constant to fixed-point array
15.	SUM	$X = \text{SUM}(X)$ $X = \text{SUM}(N)$	4513	Sum two fixed-point arrays

(6) DRI—The digital input routine, DRI, enables the experimenter to sample inputs on the DR8 and to time the interval from the function call to the response input. The statement `R = DRI(N)` clears the input register, to eliminate any premature responses, and waits up to `N` msec for one or more simultaneous input bits to be set. When the function returns, `R` represents the decimal value of the 12-bit input code. A function return with `R = 0` indicates a “dead subject” in that the subject required more than `N` msec to respond ($N < 4,096$).

(7) GRT—The GRT function may be called to recover the RT observed in a previous DRI function call. When `T = GRT(X)` is executed, `T` contains the RT in milliseconds, accurate to 1 msec. It is important that no BASIC functions requiring another overlay be executed between the DRI and GRT, since the response time is stored in the overlay area by the DRI function. The following sequence sets output Bit 10, obtains the subject's response and RT, and prints both values on the experimenter's terminal (i.e., the system console). If the subject does not respond within 4 sec, “NO RESPONSE” is sent to the experimenter and, after an additional 1-sec delay, the entire sequence is repeated.

```
100 X = DRS(1024)
105 R = DRI(4000)
110 X = DRC(1024)
115 IF R > 0 GOTO 135
120 PRINT "NO RESPONSE"
125 X = DEL(1000)
130 GOTO 100
135 T = GRT(X)
140 PRINT "RESPONSE = ", R, "R.T. = ", T
```

(8) CRI—This function enables the experimenter to retrieve single- and multiple-item character strings from the subject's terminal. The CRI function is the most complex of the general laboratory utilities and can accept string inputs in a variety of formats. The function is called by a `T = CRI(F$,N)` statement. The string argument, `F$`, specifies whether the input characters should be presented on the subject's display as they are entered. An `F$ = "ECHO"` specifies the echo mode, whereas `F$ = "NOECHO"` suppresses the echo feature (the first two characters “EC” or “NO” are sufficient to select the mode).

The numeric argument, `N`, specifies the total number of characters to be read. The function accepts fewer than `N` characters if either (1) the subject enters a return, terminating the string, or (2) the input string dimension is smaller than `N`. In the latter case, the function reads only as many characters as can be stored in the input string, up to the OS/8 BASIC limit of 80. For the CRI function, the input string is always the first string dimensioned in the BASIC experimental program.⁴ While this is admittedly an unusual way to specify the input string, unlike numeric arguments,

OS/8 BASIC handles string arguments in a highly unorthodox manner. This is the most direct way for returning string parameters from user-coded routines. If the experimenter wishes to store subject responses in an array, the fixed input string may be used to buffer incoming response strings until they can be assigned to the appropriate array after each CRI call.

Upon return, `T` contains the subject's RT in milliseconds, accurate to 1 msec, timed from the function call to the first character entry. If entry times are desired for each character in a string, then multiple-function calls are required. Because of space limitations, no “dead subject” provision is available with this function. The following sequence illustrates how the CRI function can accept two variable-length strings (up to 80 characters each) from the subject's terminal. After both strings have been entered (subject terminates each with a return), they are typed on the experimenter's console.

```
5 DIM R$(80),A1$(80),A2$(80)
```

```
100 T1 = CRI("EC",80)
105 A1$ = R$
110 T2 = CRI("EC",80)
115 A2$ = R$
120 PRINT "STRING 1:",A1$
125 PRINT "STRING 2:",A2$
```

Several points should be noted: (1) `R$` is the input string since it is dimensioned first in the program; (2) the strings are echoed as they are typed; (3) the entry latencies for the two strings are available in `T1` and `T2`, respectively (whether these are meaningful or not will depend on the experiment); and (4) the two strings are available as `A1$` and `A2$` and can be processed with any of the standard BASIC string functions (these functions will require an overlay). In the following sequence, assumed to be a later segment of the same program, the CRI function is used to record a single keypress. In the illustration, a switch is turned on via Bit 10, and the subject's single keypress and latency are recorded with CRI. Both values are typed on the experimenter's console.

```
200 X = DRS(1024)
205 T = CRI("NOECHO",1)
210 X = DRC(1024)
215 PRINT "RESPONSE = ", R$, "LATENCY = ", T
```

Signal Processing Utilities

The remaining seven user functions are referred to as signal processing utilities because they are designed to facilitate elementary signal processing operations. The standard analog-to-digital (A/D) and digital-to-analog (D/A) converters for the PDP-8/e provide either 10-

or 12-bit precision. Consequently, the following routines handle 12-bit data stored in single-precision arrays. Since OS/8 BASIC requires three 12-bit words to represent each number in floating-point format, an N-dimensional floating-point array occupies $(N*3)$ single-precision locations. OS/8 BASIC always assumes that indexing begins with zero. Therefore, the dimension statement, DIM A(1000), sets aside $1001 * 3 = 3003$ locations for the A array. For consistency, our single-precision arrays also begin indexing with zero. Two functions, FIX and FLO, are provided to transform data between the single-precision and floating-point formats. Each signal processing function requires that the user "address" a single-precision array by executing an OS/8 BASIC USE statement before calling one of the signal processing functions. For example, if a single-precision array of 3,000 elements is desired, then the array must be dimensioned to have 1,000 elements in the BASIC program [i.e., DIM A(999), since $999 + 1 = 1,000$]. If the user then wishes to operate on that array with one of the following functions, a USE A statement must be executed prior to the function call. In the following example, the function would address the 3,000-element single-precision array, A.

```
5 DIM A(999)
.
.
.
100 USE A
105 X = ZZZ( . . . )
```

As above, the variable X denotes a dummy parameter in the following detailed presentation of each function.

(9) FIX—The FIX function enables the user to convert a floating-point number to fixed-point 12-bit format and enter it into a single-precision array. Execution of $X = \text{FIX}(M,N)$ converts M to fixed point and stores it as the Nth element of the addressed single-precision array. For example, the following program segment sets up a single-precision array, A, containing the integers 1-1,000, one period of a sawtooth waveform.

```
5 DIM A(333)
.
.
.
100 USE A
105 FOR I = 0 to 999
110 J = I + 1
115 X = FIX(J,I)
120 NEXT I
```

Similarly, any complex waveform can be computed mathematically in floating-point format and prepared for output on the D/A converter (see DAC function) using this routine.

(10) FLO—This function may be used to retrieve

information from single-precision arrays and to convert it to floating-point format. The statement, $M = \text{FLO}(N)$, sets the floating-point variable M equal to the Nth element of the single-precision array addressed. In the following illustration, 1,000 fixed-point integers are recovered from the single-precision array, A, and stored in the floating-point array, B.

```
5 DIM A(333), B(999)
.
.
.
100 USE A
105 FOR I = 0 TO 999
110 B(I) = FLO(I)
115 NEXT I
```

This procedure would be used whenever sampled analog data (e.g., speech or physiological recordings) require additional processing (e.g., a fast Fourier transform).

(11) ADC—With the ADC function, the experimenter can digitize one or more points using an A/D converter and store these samples in a single-precision array. The analog input, for example, a physiological or acoustic waveform, is applied to one of the input channels of the A/D converter. The function is called with the statement, $X = \text{ADC}(M,N,C)$. The first parameter, M, is a channel/gain code for the AD01 A/D converter multiplexer. It enables the user to select the channel to sample and the amplifier gain to use (consult Digital Equipment Corporation, 1973, for details). The second argument, N, indicates the number of samples to take, and the third argument, C, specifies a sampling criterion. If sampling is to begin immediately after the function is called, then C should be zero on entry. For $C > 0$, the ADC routine will continuously monitor the selected channel until a sample value greater than C is observed. At this point, N consecutive samples are taken and stored in the addressed single-precision array. The sampling rate is determined by the rate of the system clock. In the following segment, 2,048 consecutive samples are taken on Channel 1. After sampling, the entire set of samples is converted to floating-point format for further processing.

```
5 DIM A (682), B(2047)
.
.
.
100 USE A
105 M = 1
110 X = ADC(M, 2048, 0)
115 FOR I = 0 to 2047
120 B(I) = FLO(I)
125 NEXT I
```

The ADC function may also be used to monitor analog data for a "peak" voltage to synchronize the program

with external events as long as the voltage level of the synchronizing event is known.

(12) DAC—The DAC function permits data stored in a single-precision array to be output on the AA50 D/A converter. The statement `X = DAC(M,N,T)` outputs `N` consecutive samples on Channel `M` at a rate determined by the system clock. The third argument, `T`, specifies the total number of samples to process. When `T > N`, the original `N` points are sampled repeatedly until a total of `T` samples are output. This feature enables the user to conserve storage space by storing only a single period of a periodic waveform. In this case, the periodic function is output for `T/R` sec, where `R` is the rate of the system clock in hertz. It is also possible to output only the first `F` samples of an `N`-element array by passing `F` as the second argument, `X = DAC(1,F,T)`.

The following example sets up a single 1,024-element single-precision array with linearly increasing integers between 0 and 2,046 (the largest positive voltage on the 12-bit bipolar AA-50 is produced by 2,047). Ten periods of the resulting sawtooth waveform are then output on Channel 1.

```
5 DIM A(341)
.
.
.
100 USE A
105 FOR I = 0 to 1023
110 J = I * 2
115 X = FIX(J,I)
120 NEXT I
125 X = DAC(1, 1024, 10240)
```

(13) OFF—This function enables the user to change the initial index for single-precision arrays from zero to another value. The desired value is passed as an argument in the statement, `X = OFF(I)`, where `I` represents the new initial index. Although this function may be used with any of the signal processing functions, it is particularly useful in applications in which it is necessary to isolate a segment or window of a sampled waveform. Assume, for example, that analog data have been sampled using the ADC function and are stored in a 5,000-element single-precision array. If we wish to output only the last 500 samples of these data on the D/A converter, the OFF should be called prior to DAC with an argument of 4,500. Together, the OFF and DAC functions can greatly facilitate such tasks as the segmentation and editing of digitized speech, EEG signals, or other complex waveforms.

To continue the above example, suppose that we wish to output a 500-point window beginning at the 1,024th point in the array. The following sequence illustrates how this can be accomplished.

```
5 DIM A(1666)
.
.
.
100 USE A
105 X = ADC(M,5000,0)
.
.
.
200 USE A
205 X = OFF(1024)
210 X = DAC(1,500,500)
```

In the above example, the 500-point window is output only once. If we wish to have it output repeatedly, then the third parameter to the DAC function should be some integral multiple of 500. In the following example, assumed to be a continuation of the above segment, the OFF function is used to position a 500-point window in response to starting points entered at the experimenter's console.

```
300 USE A
305 PRINT "ENTER STARTING POINT"
310 INPUT S
315 X = OFF(S)
320 X = DAC(1,500,500)
325 GOTO 305
```

Similar windowing operations can be controlled by an experimental subject by sampling keyboard or potentiometer inputs. This procedure may be useful, for example, in experiments investigating the segmentation of continuous speech or the role of duration in the perception of isolated transients (e.g., a consonant-vowel phoneme). Once changed, the initial index remains at its altered value until explicitly reset by an OFF function call, or the entire laboratory package is reloaded by the BASIC run-time system. The latter occurs whenever the user's program calls a function that resides in another system overlay.

(14) ADD—This function may be used to add a constant to one or more elements in a single-precision array. It is accessed by the statement `X = ADD(N,M)`, where `N` specifies the number of items in the array and `M` is the number to be added. Following a call to this function, `N` elements in the addressed array will be altered. It is possible for the user to increment only a portion of an array by making the first argument, `N`, smaller than the total number of samples, and/or by shifting the initial index with the OFF function. It is important to keep in mind, however, that the actual addition is performed in two's-complement arithmetic. Consequently, constants greater than 2,047 decimal represent negative numbers, and the addition of such

numbers is actually subtraction. In decimal, the two's complement of an integer between 1 and 2,047 is simply $4,096 - I$, where I represents the desired integer. If you wish to subtract 1 from all elements of a single-precision array, you call ADD with an argument of 4095. In the following example, 2 is added to all elements of a 1,024-element array.

```

5 DIM A(341)
.
.
.
100 USE A
105 X = ADD(1024,2)

```

(15) SUM—This function is used to perform an element-by-element addition of two single-precision arrays. It differs somewhat from the other signal processing utilities in that two successive function calls are required to pass the addressing information for both arrays. The first call simply passes the address of the first array; the second call passes the address of the second, target array, and a parameter indicating the total number of elements to be summed. To sum the 1,024 elements of Array A with the corresponding elements of Array B, as may be required when averaging physiological data, four statements must be executed. The results of the summation will be stored in the array addressed in the second function call, while the first array remains unchanged. The following illustrates the proper calling sequence.

```

5 DIM A(341), B(341)
.
.
.
100 USE A
105 X = SUM(X)
110 USE B
115 X = SUM(1024)

```

It is possible to sum an array into itself by omitting the USE statement before the second function call, or to sum different portions of two arrays by calling the OFF function before each SUM call. Under any conditions, the results of the summation are always stored in the addressed portion of the second, target array.

As in the case of the ADD, the SUM function effectively performs subtraction when individual elements of the addressed arrays have values between 2,048 and 4,095. Both the SUM and ADD functions can

enable the experimenter to avoid excessive and time-consuming FIX and FLO calls to perform simple additions in some signal processing applications.

IMPLEMENTATION

The entire package of 15 functions is coded in PDP-8 PAL assembler and occupies field 0 locations 3400-4577, the standard BASIC overlay area. A core image file for the package, called BASIC.UF, must reside on the system device during program execution. In addition, as illustrated in Table 3, the experimental program must also contain a UDEF statement for each routine to provide the compiler information regarding the function mnemonics and calling syntax. Before execution, the function entry points are entered into the user function table in the BASIC run-time system by using the octal debugging technique (ODT) to patch the BRTS.SV file. The octal entry point for each function is indicated in Table 2. All of the above procedures are straightforward and are well documented by Digital Equipment Corporation (1974, Chapter 6, pp. 97-100).

PERFORMANCE AND EVALUATION

After more than 3 years of almost daily use in a wide range of experimental tasks, we have been satisfied generally with the performance of this laboratory package. Technically oriented undergraduate research assistants have learned to implement relatively sophisticated experimental programs with a minimum of effort. Nevertheless, a number of limitations exist, as considered in detail below.

Sampling Rates

The sampling rate used by the DAC and ADC functions is determined by the 5-kHz DK8-EC system clock. Although this may be inadequate for many psychoacoustic studies, the user can add or substitute an additional clock that supports a faster sampling rate. While the DAC and ADC routines minimize system overhead by retaining processor control until all samples have been taken, a number of "bookkeeping" operations must be performed between samples that constrain the optimal throughput rate. The estimated optimum sampling rates under normal operation and after a minor program modification are shown in Table 4. The suggested changes simply eliminate field-boundary checking in both routines and keyboard-interrupt (via a control C) testing in the DAC routine. System operation after these changes obviously requires that addressed,

Table 3
UDEF Statements Required to Specify Mnemonics and Calling Syntax for Each Function

1	UDEF	DEL(A), DRS(A), DRC(A), CRO(A\$), CRC(A\$), DRI(A)
2	UDEF	GRT(A), CRI(A\$,A), FIX(A,B), FLO(A), ADC(A,B,C)
3	UDEF	DAC(A,B,C), OFF(A), ADD(A,B), SUM(A)

Note—See Digital Equipment Corporation (1974, Chapter 6, pp. 97-100) for a detailed discussion of the UDEF function.

Table 4
Maximum Sampling Rates (Kilohertz) under Normal Operation
and After Eliminating Keyboard and Field-Boundary Checking

Condition	Function	
	DAC	ADC
Normal Operation	14.3	34.7
Without Testing	26.0	55.5

single-precision arrays always fall within a single 4K memory field (see Digital Equipment Corporation, 1974, for a discussion of the internal representation of numeric arrays). For most applications, we have used a 12.5-kHz sampling rate without problems.

String Processing

Although the character string capabilities of OS/8 BASIC are adequate for most experimental applications, the user is restricted to uppercase characters. In addition, experiments requiring extensive on-line string processing may prove awkward using the present system. For example, if it is necessary to determine the length of a subject's string response on each trial, the BASIC LEN function must be called alternately with the CRI function. Since the two functions reside in different system overlays, program execution may prove unacceptably slow under these circumstances.

Event Timing

One of our design criteria was to achieve a timing resolution as close to 1 msec as possible. The timing capabilities actually achieved, however, deserve further discussion. During program execution, the BASIC run-time system interprets a pseudocode generated by the compiler. Consequently, every function call involves processing time that, in the case of the DEL routine, will introduce a timing error. For example, we have determined empirically that execution of the sequence, $X = \text{DRS}(Y)$; $X = \text{DEL}(N)$; $X = \text{DRC}(Y)$, results in an output of approximately $(N + 1.1)$ msec ($N > 0$). An order of magnitude reduction in timing error can therefore be achieved by calling the DEL function with an argument $M = N - 1$, where N is the desired output duration. Alternatively, sufficient unused memory locations exist in the DEL machine code to enable the user to make timing adjustments within the function itself. In the latter case, timer resolution can be improved to meet even the most extreme laboratory requirements.

In contrast to the above, output durations controlled by the DAC function are limited only by the resolution of the system clock. For example, a 5-kHz system clock enables output pulses of $(N * .2)$ msec duration (i.e., integer multiples of the clock period). However, pulse output is not immediate with execution of the $X = \text{DAC}(M, N, T)$ statement, since the system consumes some time in interpreting the function call and preparing

for output. In this case, approximately 2.5 msec elapse between the function call and pulse output.

In addition to output event timing, a second major consideration involves the timing of input events, particularly subject RTs. Since both the CRI and DRI functions retain processor control until a response is received (or, in the case of DRI, the "dead subject" interval is up), 1-msec accuracy can be obtained. Further, both functions begin execution by clearing their respective input registers to eliminate premature responses. The user should recognize, however, that a constant time interval will elapse before timing begins, since the system must process the function call. In our laboratory, constant initial delays of approximately .6 and .8 msec are observed for the DRI and CRI functions, respectively.

Input/Output

Standard OS/8 BASIC file functions may be used to transmit or receive information from any peripheral for which a device handler exists in the OS/8 system. Although the standard I/O routines were obviously not designed to handle single-precision data, they may be used for this purpose. To accomplish this, the user must use the BASIC numeric file format and read or write only one floating-point number for every three single-precision values. This capability is useful for passing single-precision data between analysis or synthesis and experimental control programs. Be forewarned that I/O is relatively slow using this method, and that the BASIC file functions require a separate overlay.

Transportability

The final issue examined concerns the transportability of the present package to other PDP-8-based laboratories. As indicated in the introduction, the routines require a minimal OS/8 system, as well as a number of laboratory peripherals. We have used the MQ register for passing arguments between subroutines, and consequently, without modification, the package requires a PDP-8/e or more recent processor. Since many laboratory peripherals are handled in a similar fashion, a user with slightly different peripheral configurations should find it relatively easy to modify the routines. Similarly, PDP-8 PAL assembler code may be added or deleted to make minor modifications for specific applications (there are approximately 23 unused locations in the overlay area). For example, a double-precision counter can be substituted for the DRI "dead subject" parameter, or the initial keyboard-clear feature of CRI function can be eliminated. It is possible also to test several subjects simultaneously (in a trial-locked fashion) with a somewhat more extensive modification. A number of utility subroutines are included in the present package, as well as in OS/8 BASIC, that should facilitate changes of this sort. The source code, which is

available from the author, is reasonably well documented and contains a table of pointer addresses to most of the OS/8 BASIC utilities.

Finally, the present package can be useful to investigators who wish to code BASIC user functions. Without the OS/8 BASIC sources (we found them to be prohibitively expensive), the user is certain to encounter a number of difficulties in interfacing assembler functions with the BASIC run-time system. Examination of the assembler code for the present package should make this task somewhat easier.

REFERENCES

- DIGITAL EQUIPMENT CORPORATION. *Small computer handbook*. Maynard, Mass: Author, 1973.
 DIGITAL EQUIPMENT CORPORATION. *OS/8 handbook*. Maynard, Mass: Author, 1974.

NOTES

1. A minor difficulty with OS/8 BASIC lies in the fact that

most powerful string and numeric functions are provided in separate overlays that cannot be core-resident with the user overlay. A complete list of the functions that require a separate overlay is found in Digital Equipment Corporation (1974, Chapter 6, p. 101). It is necessary to perform such functions at times when precise experimental control is not necessary. For example, a complete block of stimuli can be generated, randomized, and stored in arrays before beginning the experiment.

2. In systems having floating-point hardware (FPP-12 or FPP-8A) and FORTRAN IV, the user may wish to take advantage of the order of magnitude faster throughput of FORTRAN IV for off-line processing.

3. Although a 5-kHz DEC DK8-EC is used as the system clock, we use a laboratory-constructed programmable clock with computer-selectable rates (using a 16th function not described here) for signal processing applications. For generality, the present paper is written as though the DK8-EC were the only clock.

4. The user must explicitly dimension the CRI input string early in the program to avoid possible problems with implicitly dimensioned strings.

(Received for publication February 8, 1979;
 revision accepted May 15, 1979.)