Behavior Research Methods, Instruments, & Computers
1986, 18 (1), 32-35

COMPUTER TECHNOLOGY

Apple II assembly language routines for digital data acquisition and contingency control

DOUGLAS K. RUSH
Cassella AG, ZNS Pharmaforschung, Frankfurt, West Germany

Modifications and corrections to Rayfield's (1982) assembly language routines for data acquisition and experimental control using the Apple II computer are described. The modified routines, together with an optoisolated hardware interface, provide a reliable, accurate, and easy to use microcomputer system for experimental control and data acquisition.

The Apple II computer, due to its modular design, expandability, and excellent documentation, has found widespread acceptance in behavioral laboratories. Specialized systems have been designed for specific applications, such as operant conditioning (Emmett-Oglesby, Spencer, & Arnoult, 1982), classical conditioning together with measurement of physiological variables (Poltrock, Scandrett, & Gormezano, 1980), temperature biotelemetry (Cunningham & Peris, 1983), the study of stimulus control (Gordon, Foree, & Eckerman, 1983), and observational scoring of human social behavior (Flowers, 1982).

Other systems developed for the Apple II computer to control behavioral experiments are suitable for general use in practically any situation where only digital data are required. The simplest of these general purpose systems utilize the game input/output (I/O) port connected to a simple interface with programming implemented in BASIC (e.g., Bozarth, 1983). Such a system has a limited number of I/O channels and very slow execution speed.

By utilizing a combination of assembly language and BASIC programming coupled to a parallel interface card in one slot of the Apple II, a system that provides general usage in a variety of behavioral paradigms, fast execution, and a large number of I/O channels can be realized (see Mapou, Borowiec, Richards, & Byrd, 1984, for an implementation of this integrated approach on the TRS-80 microcomputer). Fast, accurate recording and control of single channels is implemented through the use of machine language bit-masking routines. Some of the assembly language routines are interrupt driven, with the counting of interrupts serving as the basis for a clock. Such systems for the Apple computer (DOS Version 3.3) are available commercially (Adalab system[1]) or are described in the literature (Rayfield, 1982).

The technical assistance of Werner Farber with the system's hardware is gratefully acknowledged. The author's mailing address is: Cassella AG, ZNS Pharmaforschung, Hanauer Landstrasse 526, 6 Frankfurt 61, West Germany.

Rayfield and Carney (1981) provided a description of a general purpose hardware/software system for controlling behavioral experiments for 6502 microprocessor-based computers. Rayfield (1982) described the system as implemented on the Apple II computer. This general-purpose digital I/O system has widespread applicability in the behavioral sciences, and it shares with the specialized systems mentioned above the combined use of BASIC and assembly language programming, the use of interrupts, and a relatively sophisticated hardware interface.

SYSTEM DESCRIPTION

The distinguishing feature of the system is the interrupt driving of assembly language routines at a constant 100-Hz rate (.01-sec resolution) through the use of a hardware/software clock. This feature provides more accurate timing than can be realized with software loops and constant and rapid polling of inputs. The hardware consists of a parallel interface card inserted in one of the slots in the Apple II.[2] The card is programmed to generate an interrupt every .01 sec. The occurrence of an interrupt stops processing of the BASIC program and starts two assembly language routines. In the first routine, counters are incremented, providing the basis for calculating time. (See Post & Fox, 1982, pp. 277-278, for a discussion of various approaches for measuring time with microcomputers, and Flowers, 1982, p. 242, for a description of interrupt-based timing.) Time in seconds is calculated in the BASIC program as described below. In the second routine, the switch inputs are sampled and saved for transfer to BASIC.

Inputs are sampled 100 times/sec (the 100-Hz interrupt rate) by an assembly language routine which reads a parallel 8-bit port and checks for changes in any of the bits since the previous read, utilizing the Boolean arithmetic operations available in the 6502 instruction set (Zaks, 1980). Changes in any of the bits, indicating the occurrence of a response, are saved (i.e., buffered) for

transfer to BASIC. The assembly language routine for transferring inputs is CALLed from BASIC and is discussed in more detail below.

The actual rate of input sampling is limited by the speed of execution of the BASIC program, which processes the inputs and controls the outputs, as only one response per input channel can be saved for transfer to BASIC at a time. High input rates by a single subject, such as a pigeon's keypecking, result in repeated inputs on the same channel. If the BASIC program executes too slowly (i.e., CALLs for an input less often than the pigeon keypecks), some of the inputs will not be recorded. The problem is compounded if several test chambers are run simultaneously.

The input (behavioral responses) and output (stimuli and reinforcers) of individual I/O channels is implemented with assembly language bit-masking routines which are transparent to the BASIC programmer.[3] If BASIC PEEKs and POKEs are used to read inputs and control outputs, as in some systems (Jenkins, 1982), relatively complicated BASIC programming is required to control the correct bits, and execution speed is markedly slowed.

INFORMATION TRANSFER FROM ASSEMBLY LANGUAGE TO BASIC

The assembly language routines described by Rayfield (1982) provided the basis for the development of our own interface system for controlling and recording behavioral experiments. In implementing this system, problems arose with two aspects of the routines. Both involved the transfer of information (inputs and time) from the assembly language routines to BASIC and resulted from the updating of information by the routines during those phases of BASIC programs in which this information was being read with PEEKs.

The first problem involved the transfer of information used to calculate the time. Four addresses (bytes) are used in the Rayfield (1982) routine, resulting in a maximum total time of over 4,678 h with a resolution of 10 msec. For our purposes, and for most applications in behavioral psychology, three addresses are sufficient. They provide a maximum cumulative time of over 18 h and result in an increase of execution speed for both the assembly language and BASIC programs.

To read and calculate the time, the following BASIC program statement is used:

```
TIME = PEAK (A1) / 100 + PEEK (A2) + PEEK (A3) * 256
```

where A1, A2, and A3 are the addresses in which the counted interrupts are stored by the assembly language program. This BASIC program statement uses the three values to calculate the number of seconds elapsed (XXXXX.XX) since the beginning of the program.

The communication problem between BASIC and assembly language as implemented by Rayfield (1982) results when an interrupt occurs during execution of the above program line, in the case where a cascade from one counter address to another is necessary because an address has reached its maximum value. (This error can also occur in the routines of Gordon et al., 1983. See p. 164 of that article.) For example, if an interrupt occurs between the PEEKs of A1 and A2, which results in a resetting of Al to 0 and an increment of A2, the time read will be .99 sec more than the actual time. A more drastic error occurs if an interrupt updates the counting addresses A2 and A3 as they are being PEEKed from BASIC; the resulting error is 255.99 sec over 4 min. Figure 1 illustrates an example of a series of PEEKs of the counting addresses before, during, and after the occurrence of an interrupt in which addresses A2 and A3 are incremented.

	A1	A2	A3	TIME (sec)
BASIC PEEK's	98	255	10	2815.98
	Interrupt - A1 incremented			
BASIC PEEK's	99	255	10	2815.99
BASIC PEEK's	99	255	Interrupt!	
	A1 and A2 reset to 0, A3 incremented to 11			
BASIC PEEK	(already read)		11	3701.99 (error)
BASIC PEEK's	0	0	11	2816.00

Figure 1. Illustration of an error in the calculated time as a result of incrementing the counting addresses during their transfer from assembly language to BASIC.

The second communication problem between the interrupt-driven assembly language routines and BASIC occurs with the PEEKing of inputs (behavioral responses). BASIC reads the value of the input with a PEEK and then signals to the assembly language program that another value can be passed by POKEing a 0 into the same location. The problem with this programming technique is that it has the same communication error noted in the transfer of the timing counters; updating of the input must be prevented during its transfer from the assembly language routine to the BASIC program. If an interrupt occurs between the execution of the PEEK and POKE commands in the case where no input is waiting from the previous interrupt (i.e., the value of the transfer address is 0) and a behavioral response has occurred between the previous and present interrupt, the input will be invalidly transferred. Under these conditions, the assembly language routine places the input into the transfer address which BASIC immediately erases by POKEing a 0. Figure 2 illustrates this sequence of events.

In improving the routines to properly transfer both the inputs and time values, the same general strategy was used. The information to be passed to the BASIC program was transferred by assembly language routines CALLed from BASIC from those addresses used by the interrupt-driven routine to addresses used by BASIC PEEKs. To prevent updating of the information during the transfer process, a flag was set indicating to the interrupt-driven routines that the routines CALLed from BASIC were in the process of transferring the information. At the conclusion of the transfer process, the flag

```
          Input address value
PEEK              3
POKE              0       valid transfer
-------------------------------------------------------
PEEK              0
     Response occurs on channel 3
     Interrupt!   3       value 3 placed into address
POKE              0       invalid transfer,
                           value 3 erased!
```

Figure 2. Sequences of events in which the input value is validly and invalidly transferred to BASIC.

was cleared and control returned to the BASIC program, which then read the desired information with a PEEK (input) or PEEKs (time). As an example of this strategy, a schematic of the programming used to input responses is shown in Figure 3.

The technique for transferring the time counters is the same, except that three addresses instead of one must be transferred. After making the changes in the timer and input transfer routines, we have had no timing problems or failures to record incoming inputs.

HARDWARE CONSIDERATIONS

To use the interrupt-driven assembly language routines described in this paper, the user needs a parallel interface card (discussed above), which plugs into one of the Apple II slots, and a hardware I/O interface (i.e., bus extension), preferably optoisolated, for driving feeders, lamps, and so forth and for recording switch inputs. As the sampling rate is increased, hardware debouncing of inputs is preferable to the software debouncing originally included in the Rayfield routines. The software debouncing was achieved by requiring that an input be present for three consecutive interrupts, that is for .02 sec, which reduced the sampling rate per channel to one half the interrupt rate. With hardware (e.g., Schmitt trigger or flip-flop) debouncing, the sampling rate is the same as the interrupt rate (within the speed limitations of BASIC discussed above). Hardware input latching, in which inputs are held until read, should be utilized if response durations less than the interrupt rate of .01 sec might occur. Several interfaces with these characteristics have been described in the literature (e.g., Falcone, Warren, & Rosellini, 1984; Jenkins, 1982). To avoid having to build the optocoupler interface for use with a parallel interface card, a commercially available unit such as the LVB interface can be purchased.[4]

In the most recent implementation of a contingency control and digital data acquisition system based on the routines of Rayfield (1982) and the corrections and modifications to those routines outlined in this paper, we employ both Apple II+ and Apple IIe computers with a parallel interface card in Slot 5 and an LVB interface card in Slot 7 connected to an LVB bus extension. The assembly language program is written to interface with a bus extension which includes up to 2 input and 4 output cards, each with 8 channels (16 inputs and 32 outputs maximum). The assembly language routines are written in a modular format using symbolic programming, thus facilitating their modification and expansion. This system is used to control and record such diverse behavioral paradigms as passive avoidance, Y-maze discrimination, and photocell activity. The same assembly language routines can be used with all apparatus and paradigms; most of the programming effort involves writing BASIC routines to control experimental contingencies and manage and summarize the collected data for statistical analysis.

AVAILABILITY

The assembly language routines described in this paper are available on disk (Apple DOS 3.3) from the author for $20. Both assembled (Apple Toolkit Editor/Assembler) and commented source versions are included. Also included is a commented BASIC program illustrating the use of these routines. Knowledge of assembly language is not necessary to use the routines with the IBS parallel interface card and LVB bus extension; for other hardware interfaces, modification of address values and/or logic may be necessary.[5]

```
BASIC program

   CALL GETINPUT : REM - ASSEMBLY LANGUAGE TRANSFER ROUTINE

   RESPONSE = PEEK (INPUT)

Routine GETINPUT

   1) Set flag to 1

   2) Transfer value at address used by interrupt driven routine to
      address used by BASIC PEEK

   3) Set address used by interrupt driven routine to 0

   4) Set flag to 0

   5) Return to BASIC
```

Figure 3. A programming schematic illustrating the technique for correctly transferring input values from the assembly language routine to BASIC.

REFERENCES

BOZARTH, M. A. (1983). A computer approach to measuring shuttle box activity and conditioned place preference. *Brain Research Bulletin*, **11**, 751-753.

CUNNINGHAM, C. L., & PERIS, J. (1983). A microcomputer system for temperature biotelemetry. *Behavior Research Methods & Instrumentation*, **15**, 598-603.

EMMETT-OGLESBY, M. W., SPENCER, D. G., & ARNOULT, D. E. (1982). A TRS-80-Based system for the control of behavioral experiments. *Pharmacology Biochemistry & Behavior*, **17**, 583-587.

FALCONE, R. V., WARREN, D. A., & ROSELLINI, R. A. (1984). Response input interface. *Behavior Research Methods, Instruments, & Computers*, **16**, 285-287.

FLOWERS, J. H. (1982). Some simple Apple II software for the collection and analysis of observational data. *Behavior Research Methods & Instrumentation*, **14**, 241-249.

GORDON, W. A., FOREE, D., & ECKERMAN, D. A. (1983). Using an Apple II microcomputer for real-time control in a behavioral laboratory. *Behavior Research Methods & Instrumentation*, **15**, 158-166.

JENKINS, W. M. (1982). Interfacing the Apple II for the behavioral laboratory. *Behavior Research Methods & Instrumentation*, **14**, 345-347.

MAPOU, R. L., BOROWIEC, F. M., RICHARDS, J. B., & BYRD, L. D. (1984). Microcomputer control of behavior and acquisition of digital and analog data. *Computer Programs in Biomedicine*, **18**, 61-76.

POLTROCK, S. E., SCANDRETT, J., & GORMEZANO, I. (1980). Microprocessor control and A/D data acquisition in classical conditioning. *Behavior Research Methods & Instrumentation*, **12**, 120-125.

POST, T. A., & FOX, J. L. (1982). Programming experiments on a TERAK 8510/A microcomputer. *Behavior Research Methods & Instrumentation*, **14**, 276-280.

RAYFIELD, F. (1982). Experimental control and data acquisition with BASIC in the APPLE computer. *Behavior Research Methods & Instrumentation*, **14**, 409-411.

RAYFIELD, F., & CARNEY, J. (1981). Controlling behavior experiments with BASIC on 6502-based microcomputers. *Behavior Research Methods & Instrumentation*, **13**, 735-740.

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

NOTES

1. An ADALAB interface card, which includes eight channels each of digital input and output, a clock, and D/A and A/D convertors, and the controlling software are available for $495 from Interactive Microware, Inc., P.O. Box 139, Dept. 1, State College, PA 16804.

2. In our system, a 6522 parallel interface card is employed (IBS Computertechnik, Bielefeld, West Germany). The parallel interface card (Model 7720, $100, from California Computer Systems, 250 Caribbean Drive, Sunnyvale, CA 94806) is compatible with the programs described in this article except for the addresses of the control register and input and output ports. (See Rayfield, 1982, for the correct address values.)

3. Bit masking is accomplished with Boolean arithmetic operations built into the 6502 microprocessor. The routines manipulate individual bits of the 8-bit input and output bytes and, thus, are able to sense the occurrence of a behavioral response and to control individual outputs. (See Zaks, 1980, for an introduction to 6502 assembly language programming and I/O interfacing.)

4. An LVB optoisolated hardware interface, including interface card, bus extension, two input and four output cards, is available from Med Associates, Box 47, East Fairfield, VT 05448 for approximately $1,850.

5. Rayfield (1982) used the parallel interface card not only as a clock, but also as an I/O port. The present routines utilize the LVB bus extension as an I/O device and the parallel card only to generate interrupts. This change necessitated alteration of the address values for the I/O ports. Also, depending upon which parallel interface card is used, changes in the address values of the registers may be necessary.

The original Rayfield (1982) routines detected a change from logic 1 to logic 0 (switch closure to ground), whereas the modified routines described in this paper for use with the LVB interface detect a change from logic 0 to logic 1. This necessitated changes in the logic programming (bit masking microprocessor operations) used to detect alterations of individual bits.

(Manuscript received August 10, 1985;
revision accepted for publication January 22, 1985.)