Behavior Research Methods, Instruments, & Computers
1984, 16 (3), 288-293

Microcontrolled stepping motors in behavior research: Two application examples

GARY P. FINLEY
University of Alberta, Edmonton, Alberta, Canada

An inexpensive microprocessor-based motor controller is described. Two examples of research applications are discussed, and a listing of a motor control program is included. Some comparisons are made between the conventional methods of laboratory motion control and the microcontrolled stepping motor approach.

Many laboratory experiments in behavior research involve the physical translation or rotation of some component of an apparatus. Traditionally, the moving parts of laboratory apparatus have been controlled by combinations of electric motors and the electromechanical hardware (such as clutches, gear trains, and limit switches) needed to mechanically program conventional ac and dc electric motors to perform specific functions. This hardware-programmed approach generally provides quick and inexpensive answers to elementary motion control requirements. However, if an experiment requires precise rotation angles or translation distances, or if an apparatus must execute a detailed series of preprogrammed motions, then the standard electromechanical approach to motion control may not be satisfactory. Complex mechanically programmed motion systems are typically difficult to design and expensive to build and require careful maintenance if they are to operate reliably.

In cases in which the mechanical demands of an experiment are difficult to meet with the electromechanical approach, a stepping motor can often provide a simple alternative solution. Stepping motors can turn through exact angular increments in either direction, producing precise bidirectional fractional rotations without the use of limit switches or reversible gearboxes. Simple lead-screw or rack-and-pinion mechanisms can convert these rotary motions into mechanical translations. Because stepping motors exert a holding torque when not rotating, they can be used to fix a component in position without the use of electromechanical clutches or brakes. Stepping motors can also be made to rotate at precise rates of speed without the feedback networks that are required to regulate the rotation speed of conventional ac and dc motors.

When a stepping motor is used in place of a mechanically programmed motor control system, the details of the motion control task are transferred from the electromechanical into the electronic domain. To make a stepping motor execute a desired sequence of motions, it is necessary to provide the motor with a series of electronic control signals that regulate the flow of current through the electromagnetic coils that turn the motor armature. In the past, it was difficult to produce an electronic control system that would cause a stepping motor to execute a particular sequence of motions. Such controllers were complex and expensive pieces of electronic hardware, and the specific repertoire of stepping motor functions provided by a given controller might not satisfy all of the demands of a particular experiment. With the advent of microcomputers, this difficulty is easily overcome. An inexpensive microcomputer-based motor control system can be made to produce a wide variety of stepping motor actions and can easily be adapted to meet the requirements of many different experimental situations.

Requests for reprints should be sent to Gary P. Finley, Department of Psychology, Biological Sciences Building, University of Alberta, Edmonton, Alberta, Canada T6G 2E9.

ELECTRICAL SIGNALS REQUIRED TO OPERATE A STEPPING MOTOR

The stepping motors used are of the Slo Syn series manufactured by the Superior Electric Corporation of Bristol, Connecticut. The motors in this series cost from $125 to over $250, depending on the amount of torque produced, the maximum step rate, and the step-angle accuracy. These motors use a set of four electromagnetic coils to turn the motor armature through a small angular increment (1.8 deg, or .005 revolution) when the control electronics makes specific changes in the current flow through these coils. Four combinations of coil current flow make up the sequence of states that cause the motor to step. The motor executes one angular step whenever the controller advances from one state to the next in the repeating four-state sequence of these coil-current combinations.

To turn the motor through a given angle, the controller issues the state changes representing the number of motor steps needed to produce the desired amount of rotation. For example, 50 steps of 1.8 deg each will turn the motor through a 90-deg angle. For continuous

Copyright 1984 Psychonomic Society, Inc.

rotation at a specified speed, the controller generates an unending sequence of motor steps separated by an appropriate time delay. Thus, with steps of .005 revolution, a speed of 1 revolution/sec requires 200 motor steps/sec. To produce this motor speed, the controller would insert a time delay of 5 msec after each motor step is executed.

Any microcomputer with at least 4 bits of parallel data output capacity can perform the simple data manipulations involved in stepping motor control. However, the electromagnetic coils that act on the armature of a stepping motor draw much more current than the logic electronics of a computer can provide. This means that an electronic interface consisting of four dc current amplifiers (one for each motor coil) is required between the low-power logic components of the computer data port and the stepping motor coils that they are to control. These current amplifiers can easily be built from standard discrete electronic components such as power transistors or the new power MOSFET devices. Figure 2a shows the design of a simple one-channel amplifier built from one logic gate and a power Darlington-type transistor. Four such amplifiers and a suitable dc power supply make up the interface needed to allow a microcomputer to drive stepping motors that draw coil currents of up to 3 A.

MICROPROCESSOR-BASED STEPPING MOTOR CONTROLLERS

In spite of their attractive mechanical features, stepping motors have little utility as components of laboratory apparatus if each motor requires the presence of an expensive microcomputer to act as its controller. Although any well-equipped laboratory computer with a coil-driving interface can be used as a stepping motor controller, the task can also be done by a single-purpose microprocessor-based controller, which can be built for a fraction of the cost of a complete laboratory computer system.

A microcomputer suitable for general laboratory purposes contains a microprocessor, memory, laboratory-data-acquisition interfaces, and the peripheral devices required to make the machine a convenient tool for the development of laboratory software. These peripherals include a keyboard, text display, mass storage device, and a printer. Each of these is connected to the system bus by an electronic interface, and each requires appropriate operating software. A computer equipped as described costs at least $2,000, but the computer components needed to run a stepping motor can be assembled into a working motor controller for about $100.

Such a controller consists of a microprocessor, some program storage memory (usually erasable programmable read-only memory, or EPROM), some read/write memory for use as temporary storage, and an input/output (I/O) controller commonly called a peripheral interface adapter (PIA). With the addition of a power supply and a clock oscillator, these components make up a simple microcomputer suitable for the data manipulation and timing tasks required of a stepping motor controller. Since the controller does not have the computing resources needed for program development, the program to perform the control functions must be written on a general-purpose software-development computer and stored in EPROM, which is then installed in the controller as its program memory.

When power is applied to the controller, the controller performs the preprogrammed functions. During the execution of this program, the controller operates the motor by means of four PIA port bits that serve as the motor coil data outputs. Other PIA port bits can serve as inputs that allow the operator to select various optional functions within the program while it is running. For example, the program can be written to allow the operator to select the motor speed, direction of rotation, or angle of turn by closing switches that are connected to PIA inputs. These bits are read by the program, and their condition determines the motor control functions to be performed.

SELECTION OF A MICROCONTROLLER

The selection of a suitable microcontroller depends on the computers available for the development of the software that the controller is to execute. It is necessary to have complete software compatibility between the development system and the controller. This software compatibility can be achieved through the use of cross-assembler programs that allow a mainframe or minicomputer to be used to produce a machine language program that can be run in a microprocessor-based controller. It is also possible to write the software with a general-purpose microcomputer that is based on the same microprocessor as that used in the controller. We have adopted the second approach, using a general-purpose microcomputer (Micro Technology Unlimited, Raleigh, NC) as our development system. This microcomputer is based on the MOS Technology, Inc., 6502 microprocessor, which is used in several popular brands of microcomputer, including the Commodore PET/CBM series and the Apple II. By selecting a 6502-based controller, we were able to ensure complete assembly language software compatibility with the development system. The controller, the John Bell Engineering, Inc. (San Carlos, CA) Model 80-153 microcomputer, consists of a 6502 microprocessor, a 6522 PIA, 1024 bytes of read/write memory, and space for a 2048- or a 4096-byte EPROM program memory. This computer board can be purchased for $110 as a finished product or for $30 as an unpopulated circuit board. By buying the bare circuit board and adding the components, a motor controller can be built for about $60. With the addition of a case, coil drivers, power supply and switches, the total cost of the completed controller is approximately $100.

If the 6502-based microcomputer or cross-assembler software needed for use with a 6502-based controller is not available, an alternative product from John Bell, Inc., offers compatibility with another widely used microprocessor, the Zilog Z-80. The John Bell 80-280 is a controller similar to the one described above, but is equipped with a Z-80 microprocessor and the companion PIA component, the Z-80 PIO. This controller is an appropriate choice for those laboratories using Z-80-based microcomputers built around the widely used S-100 bus standard.

Using a software-compatible microprocessor/controller pair, or a larger computer with an appropriate cross assembler, it is possible to produce a machine language program that will operate in the microcontroller. In order to install this program into the controller, it must first be transferred into EPROM. To do this, the user must have access to an EPROM programmer, a device that can accept the binary data of the machine language controller program and install it into blank EPROM by means of the appropriately timed high-voltage storage process. EPROM programmers and the software necessary to operate them are available as accessories for most of the common microcomputer brands. If the program thus stored does not function exactly as intended (this is often the case with computer programs), then the user will also need an EPROM eraser to remove the flawed program and make the EPROM ready for the storage of a revised version. An EPROM eraser is an ultraviolet lamp that operates through a small quartz window in the EPROM memory to allow the charge stored by the programming process to leak away, leaving the EPROM blank and ready to accept a new program. If the expenses of purchasing an EPROM programmer and eraser cannot be justified for infrequent controller program changes, it is possible to have EPROMs programmed and erased at many retail computer stores and electronic suppliers.

TWO MICROCONTROLLED STEPPING MOTOR APPLICATIONS

The first example of an application of microcontrolled stepping motors in behavior research at the University of Alberta comes from work in sensory substitution. In an investigation of the ability of blind individuals to identify tactile stimuli, a commercial reading aid made by the Optacon Corporation produces vibrotactile images of ordinary printed characters. Studies of the ability of sighted and unsighted persons to become proficient Optacon users require that text images be presented in a controlled and reproducible manner in successive experimental trials. This requires a mechanism that can scan text samples past the Optacon pickup at a variety of accurate speeds. Optacon Corporation makes a mechanical line scanner intended for training purposes, but this simple device is not capable of the scan-speed accuracy and repeatability necessary for research. In designing a motorized text-scanning mechanism suited to these experiments, our goal was to produce accurate scan speeds over a wide speed range, from the very slow speeds required by novice Optacon users to the high speeds that can be attained by the most proficient experienced users of the device. The design goal of 1% speed accuracy over a speed range of two orders of magnitude would have been difficult and expensive to reach with conventional motor speed control techniques. The stepping motor approach led to a simple and inexpensive device that easily meets this requirement.

The text scanner for this task uses a small stepping motor to turn a 6-in.-diam drum that carries the typed text samples past the Optacon pickup. The drum is driven by the stepping motor through a pair of gears, which gives an overall motion of the edge of the drum of 0.2 mm for each motor step. This mechanical transfer function makes the text motion produced by each motor step small enough to mask its discrete nature.

The motor controller used in this example consists of a John Bell 6502 computer board, a coil-driving interface like the circuit shown in Figure 2a, a power supply, and nine switches for operator control inputs. Two momentary-contact push-button switches are used to start and stop the paper drum. These are connected, respectively, to the RESET and NMI interrupt inputs of the 6502 microprocessor. Upon receipt of a RESET input from the START button, the 6502 begins executing the program that is at the address indicated by the RESET vector. This RESET vector points to the beginning of the motor control program, and both the vector and the program itself are permanently stored in the controller's EPROM program memory. The NMI interrupt produced by the STOP button causes the 6502 to begin executing the program section addressed by the NMI vector. Like the RESET vector, the NMI vector is stored in EPROM. Following an NMI interrupt, the interrupt servicing program simply removes power from the stepping motor coils and begins executing an infinite empty loop. This condition of the controller allows manual rotation of the paper drum by the operator for changing of the source text sheets. The controller stays in this idle state until it next receives a RESET input from the closure of the START switch.

The remaining seven switches allow the operator to select the rate of text scanning by choosing the appropriate paper drum rotation speed. These switches control the condition of 7 bits of 6522 port A. Each of these 7 bits is given a default logic "1" condition by a 1,000-ohm connection to the 5-V power supply, but any bit can be put into the logic "0" state by closing the normally open switch between the bit and ground. The binary data from the port A bits under the control of these switches are used within the program as pointers to the contents of a table of values stored with the program in EPROM. The values associated with each switch setting are loaded into a counter in the 6522 PIA

to control the duration of the time delay inserted after each motor step. The delay values stored in the table are arranged so that the seven switches represent a binary-weighted indicator of the paper scan speed in millimeters per second. The switches are labeled with the numbers 1, 2, 4, 8, 16, 32, and 64. The operator can obtain any paper scan speed between 1 and 100 mm/sec by selecting the combination of switches whose labels add up to the desired speed. For example, to set the paper scan speed to 24 mm/sec, the operator would select the switches labeled 16 and 8.

The Slo Syn M061 motor used has a maximum step rate of 500 steps/sec. This means that with 0.2 mm of paper motion per step, the maximum paper speed that can be produced is 100 mm/sec. This speed is fast enough to accommodate the tactile reading speeds of the most accomplished Optacon users.

The second example of a local application of a microcontolled stepping motor comes from fatigue research with laboratory rats using a motor-driven activity wheel. This technique was employed by Premack (1971) using a wheel powered by an electrical and hydraulic mechanism. The task involves rotating an activity wheel containing a running rat at a predetermined speed until the rat becomes too tired to keep up with the wheel's rate of rotation. When the rat cannot keep up, its retrograde motion within the turning wheel causes its hindquarters to break the light beam of a lamp/photocell combination, which removes power from the driving motor and allows the wheel to coast to a stop. After a predetermined period of rest, the motor is turned on again, forcing the rat to run until the next time that it is unable to keep up with the wheel. An additional complication to this motor-control task is added by the requirement that the rat not be subjected to large angular accelerations as the wheel is spun up from rest to its final speed. The acceleration applied to the wheel must not exceed the rate at which the rat is able to increase its running speed, allowing it to stay in position at the bottom of the wheel during the acceleration phase. Obviously, this acceleration limit complicates the design of any motor-speed control mechanism that is to govern the wheel-driving motor.

In our verson of the apparatus for these experiments, the activity wheel is turned by a large stepping motor (a Slo Syn M093) that is operated by a microcontroller very similar to the one described above. The START and STOP push buttons operate in the same way as above, and eight switches are used to select among eight possible wheel speeds ranging from 5 to 40 revolutions/min (rpm) in increments of 5 rpm. The motor speed control program includes an acceleration routine that increases the rotation speed of the wheel smoothly from rest to the selected speed. The control program for this application also includes the ability to respond to one other external event, namely, the inability of a tired rat to keep up with the turning of the wheel. This response is accomplished by placing the 6502 IRQ interrupt under the control of the photodetector that identifies when the rat is tired. In response to an IRQ interrupt, the motor control program performs a rest cycle that consists of allowing the wheel to coast to a stop and giving the rat a period of rest before beginning the acceleration phase of the next period of forced exercise.

In the current version of the program, the rate of acceleration and the duration of the rest period are both fixed, but it would be a simple matter to rewrite the program to allow either (or both) of these variables to be controlled by inputs from the operator in the same way that the rotation speed is controlled in the present version.

The idea of adding additional functions to the task assigned to a small dedicated controller raises the question of the functional capacity of such devices. How complex a task can be undertaken with this approach? The main factors that limit the capacity of small inexpensive controllers are the amount of program memory available and the amount of I/O port space that the controller has for its interface to external devices. In the above examples, the programs are only 300 to 400 bytes in length, about 10% of the 4,096-byte capacity of the EPROM program memory. Obviously, much more complicated control tasks could be programmed without any need for an increase in the capacity of the controller's memory.

Although the controllers used in the examples have spare program memory available, the example tasks have nearly exhausted their I/O capacity. Of the 16 bits available in the 6522 PIA, 4 are used as motor data outputs, 7 or 8 are allotted to switch inputs, and another 2 are used by the interconnection of the timers in the 6522. This means that, in the activity wheel application (with eight switch inputs), all but 2 of the available 16 bits are in use. A more ambitious project, such as the simultaneous control of several motors, would require a controller with more I/O capacity. This is not a problem, because there are several commercial controllers available that have the added computing resources that would be needed for more complex applications. John Bell Engineering makes a larger controller board (Model 81-260) that is similar to the 80-153 used in these examples but with the addition of a second 6522 PIA, which gives it a total of 32 bits of I/O port space. The 81-260 controller also has 2,048 bytes of read/write memory, twice as much as the 80-153 board. For still larger tasks, the John Bell JBE 1 computer board has 64 bits of I/O and up to 16,384 bytes of program memory. Cubit, Inc., of Mountain View, California, and Wintek Corporation of Lafayette, Indiana, among others, make comparable high-capacity controllers that use microprocessors such as the Motorola 6809E and the Intel 8085.

DIAGRAMS OF AN EXAMPLE CONTROLLER

Figure 1 is a block diagram of the 80-153 microcomputer configured as a stepping motor controller for

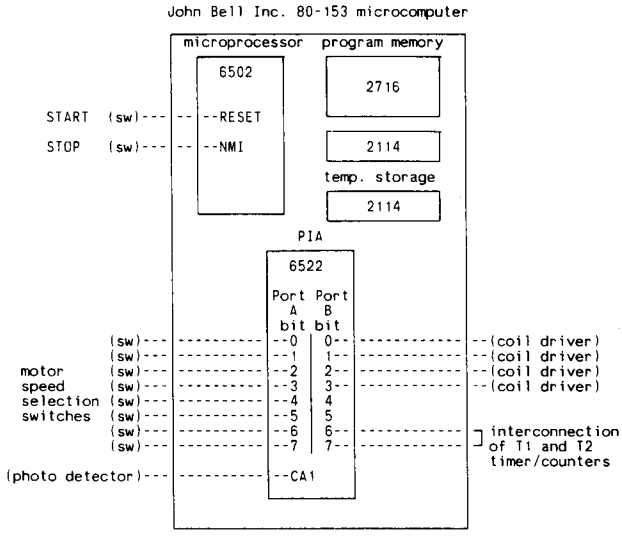

Figure 1. A block diagram of the John Bell, Inc., 80-153 microcomputer configured as a stepping motor controller for the rat activity wheel application. The details of the external components needed for the coil drivers, operator switches, and photocell rat detector are shown in (a), (b) and (c) of Figure 2, respectively.

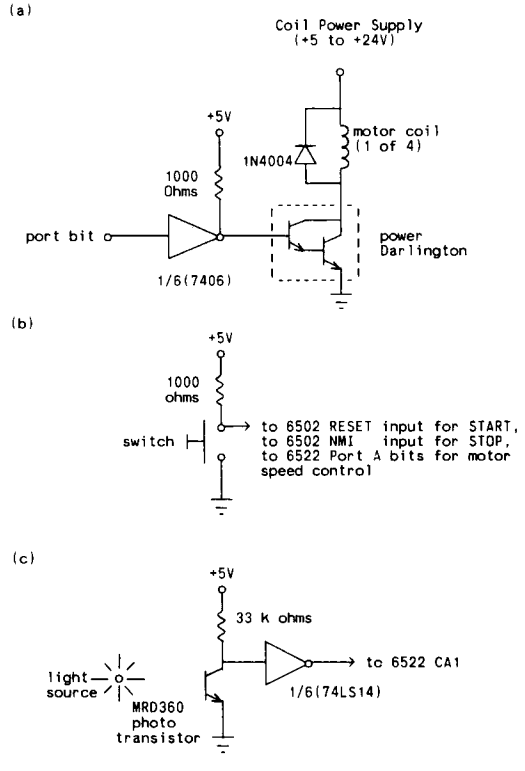

Figure 2. (a) The circuit of one of the coil drivers of Figure 1. For the M061 motor used in the Optacon application, the Darlington is a 2N6037. To handle the larger current demands of the M093 motor in the rat activity wheel application, a 2N6055 is required. (b) The circuit of the operator-input switches. For the START and STOP inputs, the switches are momentary-contact push buttons. For the motor speed control inputs, the switches are single-pole, single-throw toggles. (c) The details of the photocell detector used to generate an interrupt when the rat cannot keep pace with the activity wheel.

the activity wheel application described above. A diagram for the Optacon application differs only in the number of speed control switches used (seven instead of eight) and in the absence of the photocell circuit. The components of the motor-coil-driving interface, operator-input switches, and the photocell circuit are indicated by labels in Figure 1 and are shown in detail in Figure 2.

AN EXAMPLE PROGRAM

Listing 1 shows an example of a stepping motor control program written for the John Bell 80-153 microcomputer. This is a simplified version of the Optacon text-scanning motor controller described in the first example. The motor control program serves to demonstrate the software components needed in a practical motor control application. The program turns a Slo Syn-type motor at a variety of rotation speeds under the control of three speed-selection switches that are connected to bits 0, 1, and 2 of port A in the 6522 PIA. The speed-selection codes are matched with the programmed motor speeds as shown in Table 1. An examination of the Motor Step Loop section of Listing 1 shows that a continuous sequence of coil-control data is presented to the coil drivers, with each change of the data followed by a call of the DELAY subroutine. The hexadecimal data values 5, 6, A, and 9 used in the Motor Step Loop correspond (when used with inverting coil drivers) to the coil current combinations specified by the manufacturer for the clockwise rotation of Slo Syn stepping motors. The direction of rotation would be reversed if this data sequence were presented in the reverse order, namely 9, A, 6, and 5.

CONCLUSIONS

The stepping motor applications described above show that demands for precision-programmed motion control can arise in widely differing fields of laboratory behavior research. It should be clear that the generality of the stepping motor/microcontroller combination used here allows very similar hardware to be applied to a wide variety of motion control needs. Although the two example applications arise from very different areas of research, the electronic hardware needed for these two tasks is nearly identical. The only significant difference

Table 1
Speed Selection Inputs

	Port A Bit: 0	1	2	Motor Speed Produced:
Bit Condition:	0	0	0	0 (stopped)
	1	0	0	1 RPM
('0' = ground)	0	1	0	2 "
('1' = + 5 volts)	1	1	0	4 "
	0	0	1	8 "
	1	0	1	16 "
	0	1	1	32 "
	1	1	1	64 "

between the two motor controllers is the contents of the EPROM program memory.

If a suitable software-development computer is available, microcontrolled stepping motors can provide general-purpose programmable motion control for a broad range of experimental demands at moderate cost.

Listing 1
A Stepping Motor Control Program Example

```
;This program for the John Bell Inc. 80-153 microcomputer accepts input
;from three switches connected to 6522 port A bits 0 - 2. The binary value
;(from 1 to 7) produced by the condition of these switches is used to
;address the contents of the data array TABLE.  The values in TABLE are
;used to produce inter-step delay times that produce the motor speeds 1, 2,
;4, 8, 16, 32, and 64 RPM for switch-input binary data of 1, 2, 3, 4, 5, 6,
;and 7. The motor will start turning when a 6502 RESET is received from the
;START button if the speed-select switches are set to any non-zero binary
;speed code.  The motor stops on an NMI interrupt from the STOP button, or
;whenever all 3 speed-select switches are set to the logic '0' position.

;Assembler equate directives used to assign mnemonic labels to the addresses
;of the registers in the 6522 PIA on the John Bell computer board:

PORTA = $1C01                   ;Port A data register
DDRA  = $1C03                   ;Port A data direction register
PORTB = $1C00                   ;Port B data register
DDRB  = $1C02                   ;Port B data direction register
T1L   = $1C04                   ;Timer 1  low-byte register
T1H   = $1C05                   ;Timer 1 high-byte register
T2L   = $1C08                   ;Timer 2  low-byte register
T2H   = $1C09                   ;Timer 2 high-byte register
ACR   = $1C0B                   ;Auxiliary control register
IFR   = $1C0D                   ;Interrupt   flag  register

;Equate directive to assign the value F800 (hex) to the program counter (*).
;This sets starting address of program to John Bell EPROM socket address.

 *=$F800

;Start of program code. The following lines initialize the 6502:

START   CLD                     ;Clear the decimal mode
        SEI                     ;Disable IRQ interrupts
        LDX #$FF
        TXS                     ;Set the stack pointer to $FF

;These lines initialize the 6522 PIA:

        LDA #$00                ;Make the bits of port A inputs to be used
        STA DDRA                ;to read the speed control switches

        LDA #$8F                ;Make port B bit 7 an output and bit 6 an
        STA DDRB                ;input for interconnecting timers T1 and T2,
                                ;port B bits 0 - 3 are outputs for coil data

        LDA #$E0                ;Set the ACR register for timer 1 oscillator
        STA ACR                 ;mode, and timer 2 pulse counting mode

        LDA #03                 ;Set the registers of timer T1 to make it
        STA T1L                 ;operate as a 100 KHz oscillator.  This
        LDA #00                 ;and the T1 -> T2 connection will cause
        STA T1H                 ;the T2 count to decrement every .01 msec.

        LDA #$0F                ;Set port B bits 0 - 3 high to disable the
        STA PORTB               ;(inverting) coil drivers.  This turns off
                                ;power to the motor coils initially.

 ;Motor step loop. The following code issues the four motor-coil data states
 ;in a continuous repeating sequence. Each change of the motor coil data
 ;in port B causes the coil current changes (via the coil-driving interface)
 ;that produce one motor step.  Each step is followed by a jump to the delay
 ;subroutine which determines (based on the speed-select switch settings)
 ;the duration of the time delay that is inserted between motor steps.

LOOP    LDA #$05                ;Send motor first-state data to coil
        STA PORTB               ;drivers via 6522 port B bits 0 - 3,
        JSR DELAY               ;and call the delay subroutine.

        LDA #$06                ;Send motor second-state data to coils
        STA PORTB               ;and delay as above.
        JSR DELAY

        LDA #$0A                ;Send motor  third-state data to coils
        STA PORTB               ;and delay as above.
        JSR DELAY

        LDA #$09                ;Send motor fourth-state data to coils
        STA PORTB               ;and delay as above.
        JSR DELAY

        JMP LOOP                ;Now loop back, repeating the four step
                                ;sequence indefinitely.

;Delay subroutine: This routine reads 6522 port A to obtain the setting
;of the speed-select switches connected to bits 0 - 2.  The value
;obtained from these switches is doubled to make it into an index of
;the appropriate 2-byte value in TABLE, which contains the count that
;must be loaded into the timer T2 registers to produce the inter-step
;time delays appropriate to the specified motor speed.

DELAY   LDA PORTA               ;Read the speed select switch data.
        AND #$07                ;Mask off the unused bits (3 - 7).
        BEQ STOP                ;A zero here means that no speed select
                                ;switches are in the '1' position. In this
                                ;case, do the STOP routine.

        ASL A                   ;Double the value read from speed select
                                ;switches to make it into an index into the
                                ;2-byte table of time delay values.

        TAY                     ;Put the adjusted index into the Y register.

        LDA TABLE,Y             ;Get the low byte of the delay-count.
        STA T2L                 ;Load it into the T2L register.
        INY
        LDA TABLE,Y             ;Get the high byte of the delay-count.
        STA T2H                 ;Load it into T2H.

;This loading of the T2 registers starts T2 counting down from the loaded
;value under the control of the T1 oscillator.  T1 has been set to run at
;100 KHz, so the count in T2 is decremented every 10 microseconds.

        LDA #$20                ;Mask for the IFR register T2 flag bit.
WAIT    BIT IFR                 ;Test IFR contents with this mask.
        BEQ WAIT                ;Loop, waiting for the T2 flag bit in IFR.
                                ;(This flag will be set when the count in
                                ;the T2 registers reaches zero.)

        LDA T2L                 ;This is a dummy read of the T2L register
                                ;used to clear the IFR register T2 flag.

        RTS                     ;Return to the main program when the time
                                ;delay has elapsed.

;Stop section:  This section is executed when an NMI interrupt is caused by
;the STOP button.  It is also executed if the START button is pushed when
;all of the speed-select switches are in the '0' position (speed = zero).

STOP    LDA #$0F                ;Set port B bits 0-3 to disable the coil
        STA PORTB               ;drivers.

        JMP STOP                ;Now wait here in an infinite loop.
                                ;This section can only be exited by a RESET
                                ;input from the START button.

;The following table contains the counts of 10 microsecond 'ticks' of
;the T1 clock oscillator that must be loaded into the T2 pulse counter
;to produce the inter-step time delays appropriate to the desired
;motor speeds of 1, 2, 4, 8, 16, 32, and 64 RPM. The first table entry
;($FFFF) is a dummy value which is used to put the following values
;into the appropriate places for access by the table index which is
;generated from the speed-select switch inputs.

TABLE       .WORD $FFFF, $7530, $3A98, $1D4C, $0EA6, $0753, $03AA, $01D5

;Inter-step time delay:   300,   150,    75, 37.50, 18.75,  9.38,  4.69
;(milliseconds)

;Motor speed (RPM):         1,     2,     4,     8,    16,    32,    64

;The following are the vectors for the NMI and RESET 6502 interrupts:
;The 6502 jumps to the addresses stored at FFFA,B and FFFC,D (hex)
;when these interrupts occur, so the addresses assigned to the STOP and
;START labels must be stored into these two special EPROM locations.
;This is accomplished by setting the program counter with the following
;equate directive, and issuing .WORD directives to initialize the
;appropriate locations with the addresses of the START and STOP labels.

 *= $FFFA

 .WORD STOP                     ;STOP  address stored in NMI   vector

 .WORD START                    ;START address stored in RESET vector
```

REFERENCE

PREMACK, D. (1971). Catching up with common sense: Reinforcement and punishment. In R. Glaser (Ed.), *The nature of reinforcement* (pp. 121-150). New York: Academic Press.

(Manuscript received March 21, 1984;
revision accepted for publication May 8, 1984.)