RMV ELECTRONICS INC. **Application Note:**	**Application #** 00022
	Date: May 1995
Description: Controlling Servos with an ITC-232A	**Status:** Final Version

It is possible to use an ITC232-A to control a commercial standard servo mechanism such as the ones used in remote control model planes.

These servos have 3 cables coming out: One is red and goes to +5V, the adjacent one is ground and the third one is for control. The control line requires a pulse every 25 ms. Even though this cadence is not critical, the pulse must be 0.5 ms to 1.5 ms long. One ms long pulses results in the servo going to the mid position.

Inside the servo, there is a potentiometer moved by the shaft. This pot creates an internal signal that is compared to the external pulse. If the internal and external pulse are different then the motor turns thus moving the pot which modifies the internal pulse until the length of both of the internal and external pulses are the same.

The system is very efficient for radio control since in the transmitter, a pot attached to a 555 timer is all that is required for moving the servo to any given position. We could use the ITC232-A's PWM pin to produce the control pulses but this would not yield a good resolution, nor would it allow to control more than 1 servo. Instead, we used an Intel 8254 timer/counter controlled by an ITC232-A. Please refer to the 8254 data sheet for a detailed description of this IC

Figure 1 shows the schematics. Please note that the figure assumes that all other relevant pins in the ITC232-A are connected as described in the ITC232-A User Guide, the I/O232 or I/O485 manuals, the Experimenter kit manual, etc.

Port A (PA0..PA7) works as a data bus, and Port B (PBO..PB3) works as a set of control lines. The 8254 is always selected (CS=0) Two pull-up resistors ensure that the 8254 RD and WR pins are High when PB is in a high impedance state (at power up).

This circuit can control 2 servos. Additional servos may be controlled by adding more 8254's and using additional pins from Port B or Port C connected to the 8254's CS lines to select which chip is enabled.

Should a second 8254 be used, a total of 5 servos could be controlled since the first counter inside U1 can provide the pulses source for all other sections in all 8254's.

The circuit works as follows: The frequency generated by the crystal (1) is fed into Counter 2 of the 8254. This counter is set as an 8 bit divide-by-40 counter thus generating pulses about 10us long. This means that we will be able to modify the length of the pulses generated by Counter 1 (for Servo 1) and Counter 2 (for Servo 2) in 10u intervals. This also means that the smallest number of pulses that need to be counted by Counter 1 or Counter 2 in order to produce a 0.5 ms control pulse for the servo (minimal pulse) is about 50. To get the largest pulse (1.5 ms), 150 pulses, 10uS long need to be counted. This however varies from one servo to another, and in any case the pulses from Counter 2 are 10.9u long and not exactly 10. The position of the servo can be set with a digital precision of 1 unit in 150-50 = 100 or 1%. This is good enough for most applications and probably better than the mechanical precision of any standard servo. In our experience, a difference of 1% (one unit in the value sent to PA) results in a noticeable displacement of the shaft.

Programming the 8254

This IC contains 3 counters which can be individually configured as 8 or 16 bit counters in a variety of modes. Three registers allow to write or read values in each counter (count 0, Count 1 and Count 2). A fourth register allows to program the operating modes for each counter. Address pins A0 and A1 select the register. RD and WR are used to read and write values to the various registers.

To program a counter's mode, A0 , A1 and RD must be High and CS and WR must be Low. The value to be programmed is placed on the data bus. This value is then latched in when WR is pulled back High. For a detailed insight of the registers, please refer to the heading of the example in QBasic.

In the software example, the COM port is first initialized , then the ITC232-A is put in program mode. After this, a 40Hz, 50% duty cycle signal is put out the PWM pin. This signal initiates a 0.5-1.5 ms pulse for the servo every 25 ms. Next, a little trick is used: If we configured PB first and then wrote to it, in between PB would be 0. This in turn would enable both WR and RD which is not allowed. To prevent this, we write PB **first** while the port is still in high impedance. The value is written into a "phantom" port which is then immediately placed on the pins upon configuring it as an output. Thus, we write PB first and then we configure the lowest nibble as outputs. PA is then configured as outputs, and the counters are then configured: Counter 2 is used as a square wave generator; that is,a plain vanilla frequency divider (8 bits, LSByte only). The control register value is placed on PA and then it is strobed into the control register. We then write the divider (40) into the counter (but it could be done later as we do with the other two counters). Next, we set up the other two counters in Mode 1, one-shot retriggable. This mode works as follows: When G goes high, (the PWM goes High), then the one-shot counter is triggered. Count down starts on the next CLK pulse which is generated by Counter 2. From the time Counter 2 starts counting down, OUT 2 goes low and stays that way until the counter reaches 0. Thus, OUT is Low for n number of pulses, n being the value placed in the counter. Since the count down takes less than 2 ms and each G is pulled High every 25 ms there is no timing conflict. By changing the value loaded into the counter, the pulse width on OUT can be changed. This is done by the rest of the program whenever "U" and "D" or "L" and "R" are pressed, The variables StepH and StepV define the angle of rotation each time a valid key is pressed.

Please note that the variable length pulse is Low and since it must be High for normal servo operation, an inverter is placed between the OUT pins and the servo.

PROGRAM EXAMPLE IN QBasic

```
' REMOTE CONTROL SERVO OPERATION FROM AN ITC232-A USING AN 8253 (2.6 MHz)
' OR 8254 (10 MHz) TIMER.

' This example corresponds to Application Note #22
' For clarity purposes, everything is in binary format which takes
' long to send out. To work faster translate all binary numbers into
' decimals and eliminate the spaces within the commands. You can also
' increase the baud rate using the corresponding command and then reopening
' the COM port at the right speed.

' The system uses COUNT2 in the 8254 to divide the ITC232-A clock
' frequency (3.6468 MHz) by 40.
' However, this value might need to be adjusted for different servos
```

```
' In our prototype, a value of 40 worked better.
' This 10 KHz clock is fed into the other 2 counters (COUNT0 & COUNT1)
' in the 8254 which are configured as hardware retriggerable
' one time monostables (mode 1).
' The reset is done by the PWM edge. Thus, a given number is written
' into the corresponding counter and following every raising edge of
' the PWM, the OUT pin goes Low and the timer counts down to 0. The
' OUT then goes High again and waits there until the next PWM raising edge
' comes through. Thus, the number in the counter sets the length of the
' negative pulse on OUT. Servos need positive pulses and therefore
' an inverter is used on OUT to feed the servo control line.

' CONTROL & DATA LINES:
' PB on the ITC232-A is used to control the lines on the 8254 as follows:
' ITC232-A     8254
' PB0      =>   A0
' PB1      =>   A1
' PB2      =>    W  (asserted Low)
' PW3      =>    R  (    "     " )
' CS is always to ground.
' PA is used for the data bus. Care must be taken to never have the 8254
' writing data while PA on the ITC232-A is configured as outputs since
' a collision in values will result in pins being permanently damaged.

' DESCRIPTION OF THE 8253 or 8254 REGISTERS:
' The IC has 3 counters which can be software configured as 16 or 8 bit
' counters (here we use them all as 8 bit counters).
' Each counter (COUNT0, COUNT1 and COUNT2) has a control register which
' is selected with the corresponding address on A0 and A1.
' An additional register (address 11) is used to set-up the operating mode
' for each counter independently.
' The programming sequence is to put a value on the data bus,
' then the value on the address bus (A0, A1), then pull W Low and finally
' W High again. With the ITC232-A this can be done in a simpler way:
' Since the address and the R and W lines are on the same port (PB),
' the address and W pulled Low happen at once. Pulling W High latches the
' values on A0 and A1.

' REGISTERS:
' The MODE register (accessible with A0..A1 = 11) is as follows:
' bits 7 & 6 => Counter number (0, 1 or 2 for COUNT0, COUNT1 & COUNT2)
' bits 5 & 4 => LSByte, MSByste or both. We use 01 for 8 bit counting on LSByte
' bits 3, 2 & 1 => Mode. We use mode 3 (011) for COUNT2 and mode 1 (001) for the other 2
counters
' bit 0 => binary or BCD. We alwayys use binary (0)
' NOTE: You cannot read this register back by pulling R Low.

' The register where the value to count down from is stored is
' programmed by placing the value on the data bus, selecting the A0 & A1
' address (00 = COUNT0, 01 = COUNT1, 10 = COUNT2) with W Low. When W
' goes High then counter is loaded and it starts counting down in the next
' clock edge.
```

```
' Numbers in the commands are in hexadecimal for speed and clarity


CLS
Start:
PRINT "RESET ITC232-A AND PRESS A KEY TO START"
START1: DO UNTIL INKEY$ <> "": LOOP

CLEAR
TRUE = 1: FALSE = 0

REM Open COM port
OPEN "Com1: 9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB2048,RB2048" FOR RANDOM AS #1

' Get the ITC232-A in program mode (no CR or LF) & configure it
'W$ = "CRAP": GOSUB WRITESERIAL

' Now get the PWM pin to generate the 40 Hz for the servos
W$ = "W40": GOSUB WRITESERIAL

' By placing the value out first and then configuring PB we make sure no
' collisions occur.
W$ = "PWB$0F": ' $0F=b00001111,  R & W disabled, address b11"
GOSUB WRITESERIAL

W$ = "PCB$0F": ' $0F=b00001111, Configure PB0..PB3 as outputs
GOSUB WRITESERIAL

W$ = "PCA$FF": ' $FF=b11111111, Configure PA as all outputs
GOSUB WRITESERIAL

' Set-up COUNT2 as a square wave generator (divide clock by 40).

' DATA:
' Count bits (7,6) = 10 for COUNT2, bits 5,4 = 01 for LSByte,
' mode bits 3, 2, 1 = 011 (mode 3, square wave) & bit 0 = 0 for binary.
W$ = "PWA$96": ' $96=b1001 0110
GOSUB WRITESERIAL

' CONTROL:
' R = 1, W = 0 , A0..A1 to 11 to select CTRL register
W$ = "PWB$0B": ' $0B=b0000 1011, place PA in the register with W=0
GOSUB WRITESERIAL
W$ = "PWB$0F": ' $0F=b0000 1111, latch PA into 8254 by pulling W = 1
GOSUB WRITESERIAL

'Now place the divider (40) into COUNT2
' DATA:
W$ = "PWA 40":     ' In decimal it is clearer
GOSUB WRITESERIAL
' CONTROL:
W$ = "PWB$0A": ' $0A=b0000 1010, write PA to counter 2 with W=0 & A1, A0=10
GOSUB WRITESERIAL
```

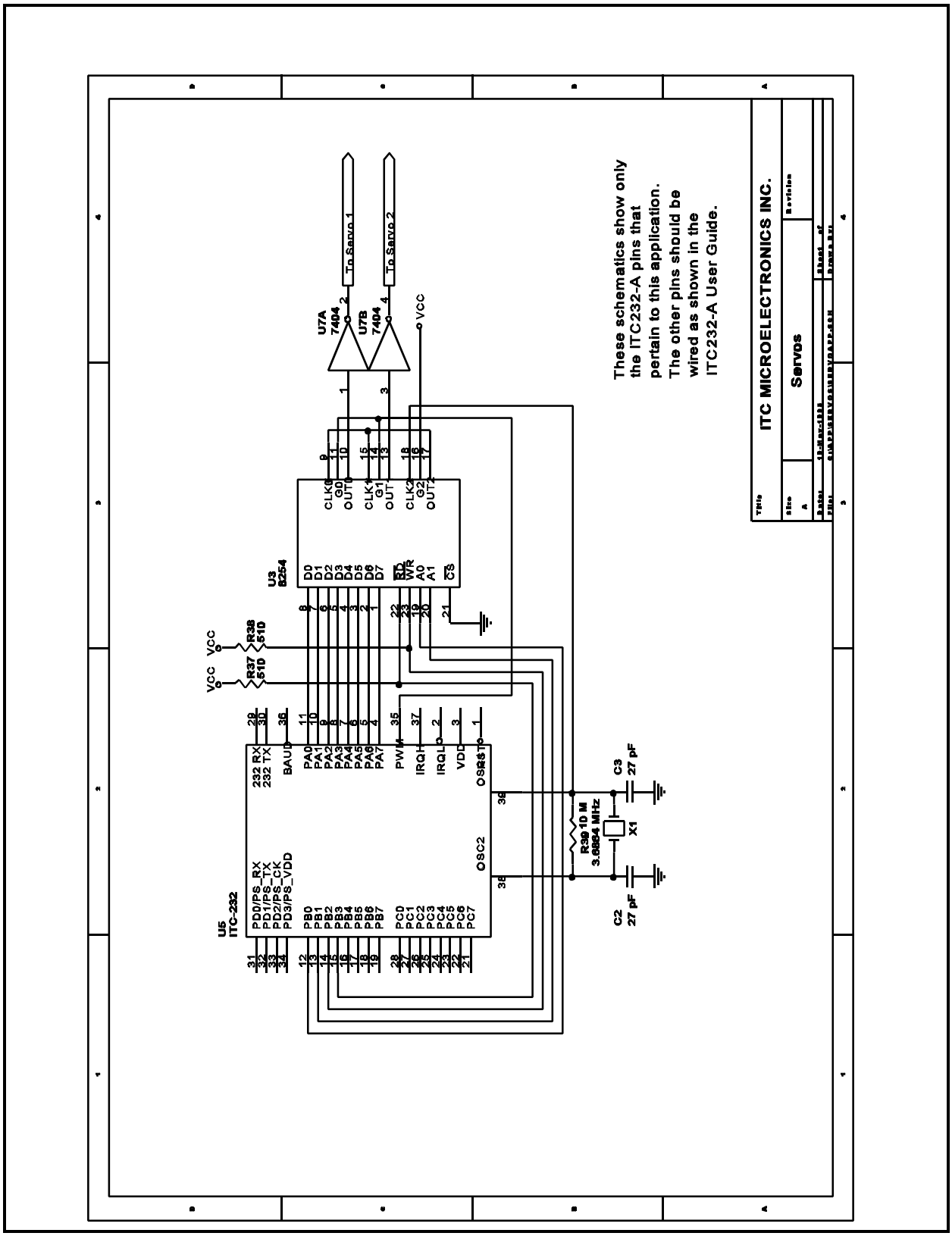
```
W$ = "PWB$0E": ' $0E=b0000 1110, latch PA into 8254 pulling W = 1
GOSUB WRITESERIAL


' Set-up COUNT0 as 1 time retriggerable. This will be the Left/Right servo

' DATA:
' Count bits (7,6) = 00 for COUNT0, bits 5,4 = 01 for LSByte,
' mode bits 3, 2, 1 = 001 (mode 1, one shot) & bit 0 = 0 for binary.
W$ = "PWA$12": ' $12=b0001 0010
GOSUB WRITESERIAL

' CONTROL:
' R = 1, W = 0 , A0..A1 to 11 to select CTRL register
W$ = "PWB$0B": ' $0B=b0000 1011, place PA in the register with W=0
GOSUB WRITESERIAL
W$ = "PWB$0F": ' $0F=b0000 1111, latch PA into 8254 by pulling W = 1
GOSUB WRITESERIAL

' A value must now be written for the counter, we use 127 (mid scale)
' DATA:
W$ = "PWA127": 'decimal is easier to use here
GOSUB WRITESERIAL
' CONTROL:
W$ = "PWB$08": ' $08=b0000 1000, writes PA in COUNT0 register with W=0
GOSUB WRITESERIAL
W$ = "PWB$0C": '$0C=b0000 1100, latches PA in COUNT0 by pulling W=1
GOSUB WRITESERIAL


' Set-up COUNT1 as 1 time retriggerable. This will be the Up/Down servo

' DATA:
' Count bits (7,6) = 01 for COUNT1, bits 5,4 = 01 for LSByte,
' mode bits 3, 2, 1 = 001 (mode 1, one shot) & bit 0 = 0 for binary.
W$ = "PWA$52": ' $52=b0101 0010
GOSUB WRITESERIAL

' CONTROL:
' R = 1, W = 0 , A0..A1 to 00 to select CTRL register
W$ = "PWB$0B": ' same as above
GOSUB WRITESERIAL
W$ = "PWB$0F": ' same as above
GOSUB WRITESERIAL

' We are done configuring. Now to make the servos work we have a loop
' where U (up) and D (down) move 1 servo and R (right) and L (left) move
' the other. Esc ends the program. Due to minor differences from one
' servo to another, we use MaxV, MinV and MaxH, MinH to set the excursion
' limits in the vertical and horizontal servos.

MaxV = 255: MinV = 0: 'Change according to your servo requirements
MaxH = 255: MinH = 0: '  "       " ...
```

```
PosV = INT((MaxV + MinV) / 2): 'Start inbetween
PosH = INT((MaxH + MinH) / 2)  'Start inbetween
StepH = 10: 'Size of motion whenever a key is asserted
StepV = 10


CLS
PRINT "USING AN ITC232-A TO CONTROL REMOTE CONTROL SERVOS"
PRINT
PRINT "U = Up, D = Down, L = Left, R = Right, Esc to exit"

LOCATE 5, 1: PRINT SPACE$(50): LOCATE 5, 1
PRINT "Vertical = "; PosV, "Horizontal = "; PosH

DO
 K$ = UCASE$(INKEY$)
 SELECT CASE K$
  CASE "U"
    PosV = PosV + StepV
    IF PosV > MaxV THEN PosV = MaxV

' The following line updates the width of the pulses. We repeated it
' at each instance, rather than creating a subroutine for clarity purposes.
' However you can set PosV and do GOSUB VServo

    W$ = "PWA" + STR$(PosV)
    GOSUB WRITESERIAL
    W$ = "PWB$09": ' $09=b0000 1001, PA in (see above)
    GOSUB WRITESERIAL
    W$ = "PWB$0D": ' $0D=b0000 1101", latch PA into 8254 pulling W = 1
    GOSUB WRITESERIAL

  CASE "D"
    PosV = PosV - StepV
    IF PosV < MinV THEN PosV = MinV

' The following line updates the width of the pulses. We repeated it
' at each instance, rather than creating a subroutine for clarity purposes.
' However you can set PosV and do GOSUB VServo

    W$ = "PWA" + STR$(PosV)
    GOSUB WRITESERIAL
    W$ = "PWB$09": ' $09=b0000 1001, PA in (see above)
    GOSUB WRITESERIAL
    W$ = "PWB$0D": ' $0D=b0000 1101", latch PA into 8254 pulling W = 1
    GOSUB WRITESERIAL

  CASE "R"
    PosH = PosH + StepH
    IF PosH > MaxH THEN PosH = MaxH

' The following line updates the width of the pulses. We repeated it
' at each instance, rather than creating a subroutine for clarity purposes.
```

```
' However you can set PosH and do GOSUB HServo

    W$ = "PWA" + STR$(PosH)
    GOSUB WRITESERIAL
    W$ = "PWB$08": ' $08=b0000 1000
    GOSUB WRITESERIAL
    W$ = "PWB$0C": ' $0C=b0000 1100
    GOSUB WRITESERIAL

  CASE "L"
    PosH = PosH - StepH
    IF PosH < MinH THEN PosH = MinH

' The following line updates the width of the pulses. We repeated it
' at each instance, rather than creating a subroutine for clarity purposes.
' However you can set PosH and do GOSUB HServo

    W$ = "PWA" + STR$(PosH)
    GOSUB WRITESERIAL
    W$ = "PWB$08": ' $08=b0000 1000
    GOSUB WRITESERIAL
    W$ = "PWB$0C": ' $0C=b0000 1100
    GOSUB WRITESERIAL

 END SELECT

 IF K$ <> "" THEN
  LOCATE 5, 1: PRINT SPACE$(50): LOCATE 5, 1
  PRINT "Vertical = "; PosV, "Horizontal = "; PosH
 END IF

LOOP UNTIL K$ = CHR$(27)

TheEnd:
CLOSE #1
END


'SUBROUNTINES

HServo:
    W$ = "WL": ' Turning OFF and ON the PWM makes the first pulse always OK
    GOSUB WRITESERIAL

    W$ = "PWA" + STR$(PosH)
    GOSUB WRITESERIAL
    W$ = "PWB$08": ' $08=b0000 1000
    GOSUB WRITESERIAL
    W$ = "PWB$0C": ' $0C=b0000 1100
    GOSUB WRITESERIAL

    W$ = "W40"
    GOSUB WRITESERIAL
```

```
RETURN


Vservo:
   W$ = "WL": ' Turning OFF and ON the PWM makes the first pulse always OK
   GOSUB WRITESERIAL

   W$ = "PWA" + STR$(PosV)
   GOSUB WRITESERIAL
   W$ = "PWB$09": ' $09=b0000 1001, PA in (see above)
   GOSUB WRITESERIAL
   W$ = "PWB$0D": ' $0D=b0000 1101", latch PA into 8254 pulling W = 1
   GOSUB WRITESERIAL

   W$ = "W40"
   GOSUB WRITESERIAL
RETURN


REM  Writing to serial port
WRITESERIAL:
'PRINT W$;
PRINT #1, W$
GOSUB READSERIAL
RETURN

REM  Reading serial port
READSERIAL:
S$ = ""
IF LOC(1) = 0 THEN GOTO READSERIAL

REM Get received string into S$
Lp1:
C$ = INPUT$(1, #1)
S$ = S$ + C$
IF C$ <> ">" THEN GOTO Lp1
'PRINT S$
REM decode string (V$) and value (V)
VALIDERROR = TRUE
ERRORCODE$ = ""
V$ = ""

FOR H = 1 TO LEN(S$)
IF MID$(S$, H, 1) = CHR$(7) THEN VALIDERROR = FALSE
IF MID$(S$, H, 1) = "?" THEN ERRORCODE$ = MID$(S$, H + 1, 1)
NEXT H

IF (VALIDERROR = TRUE AND ERRORCODE$ <> "") THEN GOSUB ERRORSUB: RETURN
'IF LEN(S$) > 4 THEN V$ = left$(S$, 4,LEN(S$)): V$ = LEFT$(V$, LEN(V$) - )
'V = VAL(V$)
V$ = ""
FOR n = 1 TO LEN(S$)
x$ = MID$(S$, n, 1)
```

```
IF x$ <> CHR$(13) THEN IF x$ >= "0" AND x$ <= "9" THEN V$ = V$ + x$
V = VAL(V$)

NEXT n

RETURN

ERRORSUB:
PRINT
PRINT "Error #"; ERRORCODE$
RETURN

AKEY:
 PRINT
 DO WHILE INKEY$ <> "": LOOP
 PRINT "ENTER TO CONTINUE, ESC TO ABORT"
 i$ = ""
 DO WHILE i$ <> CHR$(13) AND i$ <> CHR$(27)
 i$ = INKEY$
 LOOP
 IF i$ = CHR$(27) THEN RUN
RETURN


WRONG:
 PRINT "Wrong value returned."
 GOTO Start
```


ITC MICROELECTRONICS INC.
Servos
These schematics show only the ITC232-A pins that pertain to this application. The other pins should be wired as shown in the ITC232-A User Guide.
U5 ITC-232
U3 8254
U7A 7404
U7B 7404
To Servo 1
To Servo 2
VCC
R38 510
R37 510
R39 10 M
X1 3.6864 MHz
C2 27 pF
C3 27 pF
232 RX
232 TX
BAUD
PWM
IRQH
IRQL
VDD
OSC1
OSC2
PD0/PS_RX
PD1/PS_TX
PD2/PS_CK
PD3/PS_VDD