EEL5666
Robot Report
Charles Parks
December 7, 2001

<u>Table of Contents</u>

Abstract

This project taught me a lot about the difficulty of designing and constructing a robot. Robot design can seem simple until a person tries to construct one. The main parts of a robot are sensors, batteries, servos, and a micro-controller. Sensors allow a robot to see the world around it. Batteries allow a robot to move around and not stop when it reaches the end of its power cord. The servos or motors propel a robot along. The micro-controller is the brain of the robot. This device along with memory holds and runs the code, which is the personality of the robot. My robot was designed to map out a room. Currently scout, my robot, has the ability to navigate around a room and avoid hitting any thing. The other parts of my robot were not possible since I was not able to interface the mouse or get usable data from my compass.

Executive Summary

Robots are the way of the future they can help people in almost any task. This paper examines an attempt at constructing a robot that can map a room. The goal for this robot was to use a set of sensors that cost less than $100.00 and still be able to map out a room. Another goal was to interface the sensors with minimal amount of additional hardware. The optical mouse was selected as the main tool to measure distances traveled by the robot. The optical mouse seam best suited for this task since it was designed to measure distance moved over a variety of different surfaces. IR range finders along with bump sensor were used to enable the robot see obstacles. The actuation for this robot was two hacked servos. The design for this robot seamed very promising and although the constructed robot does not perform the task of mapping a room yet the creator for this robot is hopeful of still accomplishing this task.

Introduction

Robots are autonomous machines that are designed to perform tasks that are impossible, difficult, or monotonous for a person to accomplish. The individual tasks a robot must perform depend upon the overall purpose of the robot.

Scout, was originally designed to map o ut a room showing the location of all obstacles in that room. Scout was designed around two main sensors (an optical mouse and an electronic compass). Sensors are only useful if they can provide accurate data in a format that can be understood by the robot. The construction of this project taught me the importance of these two things.

Integrated System

Scout will consist of four basic systems that will enable it to map out an area.

- Navigation
- Obstacle detection
- Propulsion
- Graphical display

Each of these systems will communicate will any system it needs to and will be responsible for certain behaviors.

Navigation System

The Navigation System will be responsible for determining current position and heading. This system will use an optical mouse and an electronic compass to accomplish its tasks. The optical mouse proved too difficult to interface with the 68HC11 processor in the limited time span of this project. The electronic selected for this robot gave values that did not increase linearly with change in angle. These values may be the result of magnetic fields in the room or a malfunction of the compass. The compass lacked some features (such as the ability to give accurate readings even if slightly tilted) that would have been useful for the robot.

Obstacle Detection

The Obstacle detection system will be responsible for locating all obstacle and sudden drops in the area of operation. This system will use 4 strategically place IR sensor and 2 bump sensors. The Obstacle Detection system worked well for the robot. The value each IR sensor gave for a fixed distance varied slightly. This slight variation was not much of a problem since the robot used ranges to determine if the obstacle was too close.

Propulsion

The Propulsion system will be responsible for propelling the robot in the x and y-axis. The Propulsion system must be able to make accurate 45 and 90 degree turns as well as forward and reverse movement. This system will use two independently controlled servos and two 3-inch wheels to perform its task. The propulsion system was able to turn the robot and drive it forward. The problem of this method of propulsion was that the servos were not perfectly matched and as a result the robot will tend to drift over long distances. A robot with one servo controlling the rear wheels and another servo controlling a steering wheel would be able to travel in a strait line but may lose some of its ability to turn.

Graphical Display

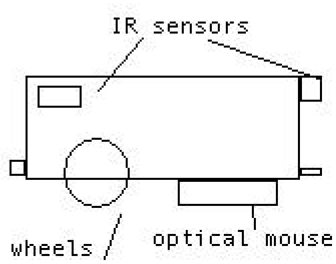
The Graphical Display system will convert the values stored in the grid into a sequence of ASCII characters and transmit them to a PC. This system will use the serial interface system of the micro controller and conversion routine to accomplish this task. The graphical system of the robot was never implemented since the mouse was never successfully interfaced. The robot did have some routines to print text on the screen and convert a binary value into an ASCII text format that could be printed to a screen. These routines were used for testing.

Mobile Platform

The purpose of the platform is to provide a place to mount the electronics for the robot. The goal of the platform I designed was to make simple shape that was symectrical about one axis. The platform should contain a way to mount the servos used to propel the robot as well as all sensors used by the robot. The two most important things that the platform should house are the micro controller, brains, and batteries, food source. An addition feature of the body that would help with debugging is external leds that indicate the mode of operation the robot is in and the status of various systems.

The placement of the optical mouse and electronic compass are critical to the success of this robot. The optical mouse needs to be place so that the y-axis it measures is the forward and reverse movement of the robot and the x-axis is the side to side movement of the robot. The optical mouse needs to be mounted level and should be as far as possible from the servos, which produce magnetic fields.

The below drawing is a preliminary drawing of the shape of the robot as seen from the side. The complete Auto-cad drawings for this robot are included in the appendices.



Actuation

Scout will be propelled and steered using two servos. The servos are basic model airplane servos that are partially hacked. The servos have been modified to allow for full 360 degrees range of motion. The modification involved removing a physically stop in the servo and disconnecting a mechanical linkage to a potentiometer in the servo. The modification converts the servo into a motor and a driver circuit. The servo is controlled by using a pulse-width-modulated signal. The width of the signal corresponds to a desired angle. When the servo is hacked the speed output of the servo is proportional to the difference between the angle given to the servo and the angle the servo thinks it is at. The servos provide a simple and reasonably priced solution to propel the robot. The turning for Scout will be achieved by turning on one servo and leaving the other servo off. The micro-controller will be responsible for turning on the servos and shutting them off when the proper amount of turning is completed. The servos in conjunction with the sensors would allow the robot to make 45 degree and 90 degree turns if the compass was working.

Sensors

Sensors enable robots to interact with their environment by providing information about that environment. The purposes of sensors on Scout are to provide it with sight, feel, location, and direction. This set of senses should allow scout to generate a map of the area it is placed in. There will be four different kinds of sensors for scout.

- 1 optical mouse
- 1 electronic compass
- 4 IR-range detectors
- 4 bump sensors

The IR – range detectors and bump sensors worked well. These sensors were easy to interface and they allowed scout to detect obstacles. The electronic compass was interfaced using input capture. The readings from the compass appeared to be non-linear these values made it impossible for me to use the compass to make accurate 90 and 45 degree turns regardless of the current heading. The non-linear values could be the result of magnetic fields in the lab or a bad calibration of the compass. The compass I selected was an inexpensive ($35.00) model from acroname.com. I am not sure if I would recommend this model to others. One of the biggest problems with this compass was that it was sensitive to tilting of the platform. There are other electronic compasses that can compensate for small degrees of tilt. The optical mouse was a necessary sensor for scout. The research I performed taught me a lot about how the data is sent to and from the mouse. I attempted to use my newly aquired knowledge to connect the mouse to the MRC11 board. The method I used was to put port D of the 68hc11 into wired-or mode and simulate through software an 11-bit SPI system. This approach appeared unsuccessful in both assembly and C. I still feel that the mouse can be connect to the 68hc11 but for now this part of the robot is unfinished. The appendix of this report shows both the information I discovered about the mouse and a block diagram for how I connected the mouse and compass to the MRC11 board.

Behaviors required by Robot

In order for Scout to perform its function it must capable of performing a series of basic behaviors.

- Detect obstacles and record their position
- Avoid obstacles in a predefined routine
- Determine current position relative to starting position
- Determine current heading
- Be able to generate and control locomotion
- Display data in a usable format

Scout had 4 main systems that it was to use to accomplish these tasks.

- Navigation
- Obstacle detection
- Propulsion
- Graphical display

Each of these systems was to communicate will any system it needs to and was responsible for certain behaviors.

Navigation System

The Navigation System will be responsible for determining current position and heading. This system will use an optical mouse and an electronic compass to accomplish its tasks. The Navigation system was unfinished in scout mainly due to the inability to communicate with the mouse.

Obstacle Detection

The Obstacle detection system was responsible for locating all obstacles in the area of operation. This system will use 4 strategically place IR sensor and 2 bump sensors. This system worked well in scout except sometimes a wheel would get stuck on an obstacle since they protruded out from the side of the robot.

Propulsion

The Propulsion system was responsible for propelling the robot in the x and y-axis. The Propulsion system must be able to make accurate 45 and 90 degree turns as well as forward and reeves movement. This system used two independently controlled servos and two 3-inch wheels to perform its task. The wheels and servos propelled the robot along as planed. The compass unfortunately was not effective in enabling the robot to make 90 and 45 degree turns.

Conclusion

This robot taught me a lot about the difficulty of interfacing different pieces of hardware. The robot's inability to communicate with the mouse prevented it from performing its primary function of mapping a room. The failure of the compass to give linear values of the degrees prevented the robot from making accurate 90 and 45 degree turns. This class was very enjoyable although stressful at times. I enjoyed helping other people with their robots and working on my robot. I consider this project in an unfinished state and plan to work on it more in the future.

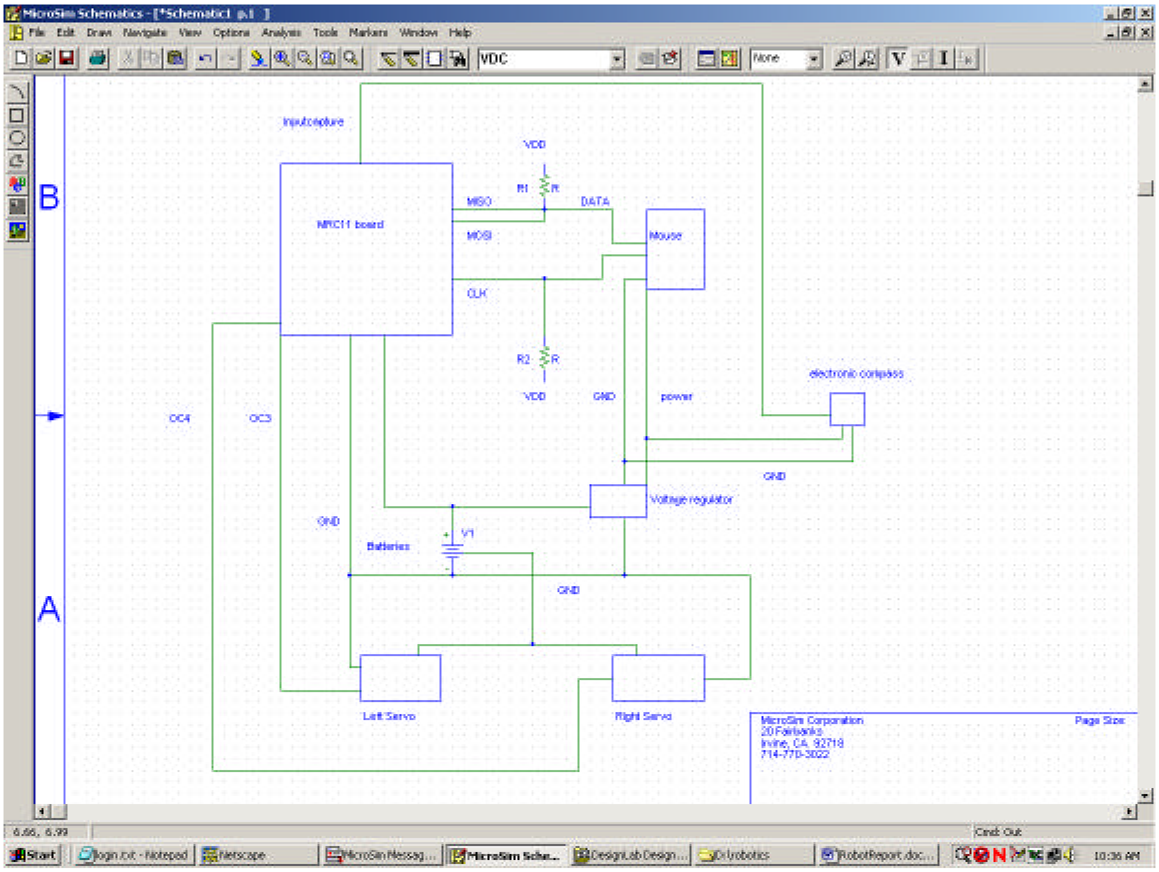
Documents

PS/2 Mouse/Keyboard Protocol, Copyright 1999 Adam Chapweske
http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm

The PS/2 Mouse Interface, Copyright 2001 Adam Chapweske
http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/mouse/mouse.html

Appendix

Block layout of circuit



Code for Scout

```
**********************************************************************
**
* Define the address locations of the various registers and user-defined
* constants used in the program
**********************************************************************
**
```

```
BASE EQU     $1000   ; base value for registers
BAUD    EQU     $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU     $102C  ; Serial Communication Control Register-1
SCCR2   EQU     $102D  ; Serial Communication Control Register-2
SCSR    EQU     $102E  ; Serial Communication Status Register
SCDR    EQU     $102F  ; Serial Communication Data Register

SPCR EQU     $0028  ; Serial Peripheral control Register
DDRD EQU     $0009  ; Data direction port D
PORTA   EQU     $0000
PORTD        EQU     $0008  ; Port D
TCTL1   EQU     $0020  ; timer control
TCTL2   EQU     $0021  ; timer control 2


PACTL   EQU     $0026   ;used to intialize RTI system
TFLG2   EQU     $0025
CFORC   EQU     $000B
OC1M    EQU     $000C
OC1D    EQU     $000D
TIC1    EQU     $0010
TMSK1   EQU     $0022
TMSK2   EQU     $0024
TFLG1   EQU     $0023
TOC1    EQU     $0016
TOC2    EQU     $0018
TOC3    EQU     $001A
TOC4    EQU     $001C
TCNT    EQU     $000E
OPTION  EQU     $0039
ADCTL   EQU     $0030
ADR1    EQU     $1031

Bit0    EQU     %00000001
Bit1    EQU     %00000010
Bit2    EQU     %00000100
Bit3    EQU     %00001000
Bit4    EQU     %00010000
Bit5    EQU     %00100000
Bit6    EQU     %01000000
Bit7    EQU     %10000000
Bits10  EQU     %00000011
InvBit6 EQU     %01111111


EOS     EQU     $04    ; User-defined End Of String (EOS) character
CR      EQU     $0D    ; Carriage Return Character
LF      EQU     $0A    ; Line Feed Character
ESC     EQU     $1B    ; Escape Charracter

** portD pin 5 and pin 3 are used to calibrate the compass

calPin  EQU    $20      ;calibration command    output
calPin2 EQU    $10      ;calibration done pin   input

**      mouse uses 2 bidirectional line to talk to host
```

```
**      mouseIn and mouseOut are tied togather with a pull up
**      1-k resistor

mouseClk        EQU    $10     ;pin 4
mouseData       EQU    $04     ;pin 2

*********************************************************
*       Compass values                                  *
*********************************************************
Max_heading      EQU     138

C_MAX   EQU     18495
C_MIN   EQU     500
C_diff  EQU     17995

*********************************************************
*       BOOLEAN VALUES                                  *
*********************************************************
TRUE    EQU     $FF
FALSE   EQU     $00

Right   EQU     $AA
Left    EQU     $55

*********************************************************
*       servo constants                                 *
*********************************************************

period  EQU     7500
LeftForward     EQU     450
LeftReverse     EQU     1050
RightForward    EQU     1050
RightReverse    EQU     450
STOP            EQU     750

*********************************************************
*       Distance constants                              *
*********************************************************
zone1   EQU     100   ; too close
zone2   EQU     75    ; visible
zone3   EQU     50    ;




***** Mouse Commands *****
Reset        EQU     $FF
Resend       EQU     $FE
Set_Defaults EQU     $F6
Disable_Data_Reporting       EQU     $F5
Enable_Data_Reporting        EQU     $F4
Set_Sample_Rate EQU  $F3   /* valid rates 20, 40, 60, 80, 100, 200
samples /sec */
Get_ID       EQU     $F2
Status_Request       EQU     $E9
Set_Resolution       EQU     $E8
```

```
** This command (Set_Scaling2) will set the mouse to 2:1 scaling
**   Mouse Counter           Reported Movement
**          0                                       0
**          1                                       1
**          2                                       1
**          3                                       3
**          4                                       6
**          5                                       9
**          N>5                                     2*N

Set_Scaling2 EQU     $E7
*** sets the scaling 1:1 recorded movement = reported movement
Set_Scaling1 EQU     $E6
requestData  EQU     $EB  // used to request mouse movement when mouse
is in Remote mode

*** Mouse modes sent as a command to mouse to set to certain modes **
Remote_mode  EQU     $F0
Wrap_mode    EQU     $EE
Reset_Wrap_Mode EQU  $EC // mouse returns to mode it was in prior to
wrap mode
Stream_mode  EQU     $EA

** commands sent from mouse to host **
Acknowledge  EQU     $FA
selftest     EQU     $AA    // means self test passed
** additionally the mouse may send the Resend Command or Error Command


************************************************************************
* Initialize Interrupt Jump Vectors
************************************************************************
        ORG     $FFFE
        FDB     Main

        ORG     $FFE2
        FDB     OC4ISR
        FDB     OC3ISR
        FDB     OC2ISR

        ORG     $FFEE
        FDB     IC1_ISR

*   (If you need to use any interrupts later,
*       put your Interrupt Jump Vectors here).
************************************************************************
*               Define Strings for displaying messages
************************************************************************
        ORG     $1040                   ;start of external memory

ClrScr  FCB     ESC,$5B,$32,$4A          ; ANSI sequence to clear screen
        FCB     ESC,$5B,$3B,$48          ;   and move cursor to home
        FCB     EOS                      ; EOS character

Prompt  FCC     @ Main Menu  @ ; Menu prompt
```

```
        FCC     @for a new BAUD rate: @ ; to explain choices
        FCB     CR, LF                  ; Carriage return and line feed
        FCC     @0=> print map @             ;
        FCB     CR, LF
        FCC     @1=> map area @
        FCB     CR, LF
        FCB     EOS                     ; EOS character

Confirm FCB     CR, LF                  ; Carriage return and line feed
        FCC     @The robot has been set to Map mode please place robot
is starting square and press rear bump sensor @

        FCB     CR, LF                  ; Carriage return and line feed
        FCB     EOS                     ; EOS character

Prompt2 FCC     @Please enter text now.@; String to prompt for text
input
        FCC     @ Or Hit ESC to show @  ;
        FCC     @BAUD menu.@            ;
        FCB     CR, LF                  ; Carriage return and line feed
        FCB     CR, LF                  ; Carriage return and line feed
        FCB     EOS                     ; EOS character

Prompt3 FCB     CR, LF                  ; Carriage return and line feed
        FCC     @Please change the @    ; String to inform users of

        FCC     @BAUD rate on your @    ; change the setting

        FCC     @computer, @
        FCC     @then hit CR.@          ;

        FCB     CR, LF                  ; Carriage return and line feed
        FCB     EOS                     ; EOS character

Prompt4 FCB     CR, LF
        FCC     @press any key test the mouse@
        FCB     CR, LF
        FCB     EOS




CompassHeading  FCB     CR,LF
        FCC     @press any key to read compass heading@
        FCB     CR,LF
        FCB     EOS


IR1     FCB     CR,LF
        FCC     @Left IR Value@
        FCB     CR,LF
        FCB     EOS

IR2     FCB     CR,LF
        FCC     @Right IR Value@
        FCB     CR,LF
        FCB     EOS
```

```
IR3     FCB     CR,LF
        FCC     @Left Front IR Value@
        FCB     CR,LF
        FCB     EOS

IR4     FCB     CR,LF
        FCC     @Right IR Value@
        FCB     CR,LF
        FCB     EOS

B1      FCB     CR,LF
        FCC     @Left Bump Value@
        FCB     CR,LF
        FCB     EOS

B2      FCB     CR,LF
        FCC     @Right Bump Value@
        FCB     CR,LF
        FCB     EOS

B3      FCB     CR,LF
        FCC     @Front Bump Value@
        FCB     CR,LF
        FCB     EOS

B4      FCB     CR,LF
        FCC     @Rear Bump Value@
        FCB     CR,LF
        FCB     EOS

SetSpeed
        FCB     CR,LF
        FCC     @Left servo @
        FCB     CR,LF
        FCC     @press 1 to increase speed @
        FCB     CR,LF
        FCC     @press 2 to decrease speed @
        FCB     CR,LF
        FCC     @Right servo@
        FCB     CR,LF
        FCC     @press 3 to increase speed  @

        FCB     CR,LF
        FCC     @press 4 to decrease speed   @
        FCB     CR,LF
        FCB     EOS



testMsg FCB     CR,LF
        FCC     @test  @
        FCB     CR, LF
        FCB     EOS

Menu    FCB     CR,LF
        FCC     @Main Menu@
```

```
        FCB     CR,LF
        FCC     @1: test servos@
        FCB     CR,LF
        FCC     @2: test Compass@
        FCB     CR,LF
        FCC     @3: test IR @
        FCB     CR,LF
        FCC     @4: test delay@
        FCB     CR,LF
        FCC     @5: test bump@
        FCB     CR,LF

        FCB     EOS


*table of shapes
SQUARE  FCB     10      ; distace
        FCB     35      ; angle
        FCB     Right   ; direction to turn
        FCB     10      ; distance
        FCB     35      ; angle
        FCB     Right   ; direction to turn
        FCB     10      ; distance
        FCB     35      ; angle
        FCB     Right   ; direction to turn
        FCB     10      ; distance
        FCB     35      ; angle
        FCB     Right   ; direction to turn
        FCB     EOS

TRIANGLE  FCB     10      ;distace
        FCB     23      ;angle
        FCB     Left    ;direction
        FCB     10
        FCB     23
        FCB     Left
        FCB     10
        FCB     23
        FCB     Left
        FCB     EOS

HOURGLASS: FCB     5     ; distance
        FCB     23    ; angle
        FCB     Left  ; direction
        FCB     10    ; distance
        FCB     23    ; angle
        FCB     Right ; direction
        FCB     5     ; distance
        FCB     23    ; angle
        FCB     Right ; direction
        FCB     10    ; distance
        FCB     23    ; angle
        FCB     Left  ; direction
        FCB     EOS
```

```
* Global Variables

delayTime       RMB
delayTime2      RMB
delayTime3      RMB

**Data            RMB     33

temp01          RMB     2
temp02          RMB     2
temp03          RMB     2
temp1           RMB     1
temp2           RMB     1
temp3           RMB     1
CNT             RMB     1
distance        RMB     1
direction       RMB     1

T_flag          RMB     1
error_flag      RMB     1




** servo duty sizes ****
** controls robot speed and direction  **
Lduty           RMB     2
Rduty           RMB     2

** Compass heading ****
** current direction the robot is pointed as read from electronic
compass **
heading         RMB     1
degrees         RMB     1       ;used when the robot is turning
new_heading     RMB     1       ;used when the robot is turning
rising_edge     RMB     2       ;used to record rising edge heading

** heading from compass = falling edge - rising edge  (pulse width)
** see documentation on electronic compass for more information


** IR / Bump sensors *************
L_IR            RMB     1
R_IR            RMB     1
LF_IR           RMB     1
RF_IR           RMB     1
F_Bump          RMB     1
B_Bump          RMB     1
R_Bump          RMB     1
L_Bump          RMB     1
toggle          RMB     1
```

```
************************************************************************
*                        MAIN PROGRAM
************************************************************************


Main    LDS     #$0041          ; Define a stack
        LDX     #BASE

        BSET    TMSK2,X Bit0    ;set the timer prescale factor
        BCLR    TMSK2,X Bit1    ;must be done in the first 64

**      JSR     InitPortD
        JSR     InitSCI         ; Initialize SCI
        JSR     Init_servos
        JSR     Init_TIC1


**      JSR     Drive2
**main2
**        LDX     #SQUARE
**        JSR     Shapes
**      BRA     main2
**        LDAA    #70
**        STAA    degrees
**        JSR     turn_left
main2
        LDX     #TRIANGLE
        JSR     Shapes
        BRA     main2
        LDAA    #70
        STAA    degrees
        JSR     turn_left
        LDX     #HOURGLASS
        JSR     Shapes
        LDAA    #70
        STAA    degrees
        JSR     turn_left


        BRA     main2


C1      LDX     #Menu
        JSR     OutStr
        JSR     InChar
        CMPA    #$31
        BNE     C2
        JSR     test2
C2      CMPA    #$32
        BNE     C3
        JSR     test3
C3      CMPA    #$33
        BNE     C4
        JSR     test5
C4      CMPA    #$34
        BNE     C5
        JSR     test4
```

```
C5      CMPA    #$35
        BNE     C1
        JSR     test6
        BRA     C1


************************************************************************
*                       SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and
receiver.
*              Effected registers are BAUD, SCCR1, and SCCR2.
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : None.
************************************************************************

InitSCI PSHA                            ; Save contents of A register
        PSHY                            ; Save contents of Y register
        PSHX
        LDX #BASE
        LDAA #$30                        ;sets Baud Rate to 9600
        STAA BAUD                       ; Set BAUD rate to 9600
        LDY  #SCCR1                     ; Load Y with address of Serial
Communication Control Register-1
        BCLR 0,Y #%11101111             ; Set SCI Mode to 1 start bit,
        BSET 1,Y #%00001100             ;  8 data bits, and 1 stop bit.
*                                       ; Enable SCI Transmitter
*                                        ;     and Receiver
        PULX
        PULY                            ; Restore Y register
        PULA                            ; Restore A register
        RTS                             ; Return from subtoutine


************************************************************************
*                       SUBROUTINE -  OutByte
* Description       : Outputs a hexadecimal number to the computer
screen
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : OutChar
************************************************************************
OutByte PSHA                         ; Save contents of A register
        LSRA                         ; shift regA to the right 4 times
        LSRA
        LSRA
        LSRA
        CMPA    #10
        BPL     letter               ; BRANCH IF PLUS
        ORAA    #$30
        BRA     out1
```

```
letter  ADDA    #$37

out1 JSR     OutChar
     PULA
     PSHA
     ANDA    #$0F
     CMPA    #10
     BPL     letter2
     ORAA    #$30
     BRA     out2

letter2 ADDA    #$37

out2 JSR     OutChar

     PULA                        ; Restore A register
     RTS                         ; Return from subtoutine

************************************************************************
*                       SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*              checking if the Transmitter Data Register is Empty
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : None.
*************************************************************************
*
OutChar PSHB                        ; Save contents of B register
Loop1   LDAB    SCSR                ; Check status reg (load it into B
reg)
        ANDB    #%10000000          ; Check if transmit buffer is empty
        BEQ     Loop1               ; Wait until empty

        STAA    SCDR                ; A register ==> SCI data

        PULB                        ; Restore B register
        RTS                         ; Return from subtoutine

*
************************************************************************
*                       SUBROUTINE -  OutStr
* Description: Outputs the string terminated by EOS. The starting
location
*              of the string is pointed by X register. Calls the OutChar
*              subroutine to display a character on the screen and
*              exit once EOS has been reached.
* Input         : Starting location of the string to be transmitted
*               : (passed in X register)
* Output        : Prints the string.
* Destroys      : regester X
* Calls         : OutChar.
************************************************************************
*
OutStr:      PSHA
OutStr1:
```

```
        LDAA    0,X                 ; Get a character (put in A
register)
        CMPA    #EOS                ; Check if it's EOS
        BEQ     Done                ; Branch to Done if it's EOS
        BSR     OutChar             ; Print the character by calling
OutChar
        INX
        BRA     OutStr1
Done:   PULA
        RTS                         ; Return from subtoutine

*
**********************************************************************
*                SUBROUTINE  -  InChar
* Description: Receives the typed character into register A.
* Input         : None
* Output        : Register A = input from SCI
* Destroys      : Contents of Register A
* Calls         : None.
**********************************************************************
*
InChar
Empty   LDAA    SCSR                     ; Check status reg.
        ANDA    #%00100000               ;     (load it into A reg)
        BEQ     Empty                    ; Check if receive buffer full
*                                        ; Wait until data present
        LDAA    SCDR                     ; SCI data ==> A register
        RTS                              ; Return from subroutine
*
**********************************************************************
*                       SUBROUTINE - SetBAUD
* Description: This subroutine changes the Baud-rate. The only effected
* register is BAUD.  If the input value is invalid, a menu/prompt is
* displayed and a new input is read.  The subroutine waits for the
* user to type a carriage return after changing the baud-rate manually
* on the PC.  It then prints out a confirmation message.
* Input         : None.
* Output        : Changes BAUD register. Repeats prompt if invalid
input.
* Destroys      : None
* Calls         : OutStr, OutChar, InChar.
**********************************************************************
*

SetBAUD PSHA                    ; Save contents of A register
        PSHB                    ; Save reg B
        PSHX                    ; Save reg X
Loop3   LDX     #ClrScr         ; Clear Screen
        JSR     OutStr          ;
        LDX     #Prompt
        JSR     OutStr          ; Print Baud-rate Menu
        JSR     InChar          ; Take menu choice from keyboard
        CMPA    #$30            ; check for unreasonable menu choice
        BLT     Loop3           ; ascii value to small to be a number
        CMPA    #$35            ; Check for unreasonable menu choice
        BGT     Loop3           ; ascii value to large to be a valid
choice
```

```
        JSR     OutChar         ; If valid input, Echo the input to
Screen

        LDX     #Prompt3        ; Inform users of change the terminal
        JSR     OutStr          ;    setting with the new BAUD rate
Wait    LDAB    SCSR            ; Check status reg (load it to B reg)

        ANDB    #%01000000      ; Check if transmit is complete

        BEQ  Wait               ; wait until TC = 1
        STAA BAUD               ; SET the new BAUD rate
Loop4

        JSR  InChar              ; Get next input from keyboard
        CMPA #CR
        BNE  Loop4              ;    wait until carriage return

        LDX     #Confirm        ; Print confirmation message
        JSR     OutStr

        PULX                    ; Restore X register
        PULB                    ; Restore B register
        PULA                    ; Restore A register
        RTS

************************************************************************
*                       SUBROUTINE - delay
* Description: This subroutine will create a delay time equal to the
value
*             stored in delayTime
*             delay = 4.5 * delayTime + 2.5  (micro-seconds)
* Input        : delayTime
* Output       : None
* Destroys     : delayTime
* Calls        : None
************************************************************************
*

delay       DEC     delayTime       ;6 cycles
            BNE     delay           ;3 cycles
            RTS                     ;5 cycles

************************************************************************
*                       SUBROUTINE - delay2
* Description: This subroutine will itialize OC2 to create a delay time
*              delay = delayTime2*(mili-seconds)
* Input        : delayTime2
* warning      : delayTime2 should be a positive integer less than 127
* Output       : set T_flag to TRUE
* Destroys     : None
* Calls        : None
************************************************************************

delay2
        PSHA
        PSHB
        PSHX
```

```
        LDX     #BASE

* clear OC2 Flag
        BSET    TFLG1,X Bit6
        CLR     T_flag
* disable OC2 output function
        BCLR    TCTL1,X Bit7
        BCLR    TCTL1,X Bit6
        LDAA    delayTime2
        LDAB    #125            ;used to convert delayTime2 into the
value
*                               ;stored for the interrupt
        MUL
        LSRD
        LSRD
        ADDD    TCNT,X
        STD     TOC2,X

* enable OC2 interrupt
        BSET    TMSK1,X Bit6

        CLI
        PULX
        PULB
        PULA
        RTS

************************************************************************
*       Interrupt Service Routine: OC2ISR                              *
*       Function:       set T_flag to true                             *
*       Input:          none                                           *
*       Output:         set T_flag                                     *
*       Calls:          none                                           *
*       Destroys:       none                                           *
************************************************************************

OC2ISR
        LDX     #BASE
        BRCLR   TFLG1,X Bit6 end_OC2ISR   ;Igonore Illegal Interrupt


        LDAA    #Bit6
        STAA    TFLG1,X ;clear Flag
        LDAA    #TRUE
        STAA    T_flag

* disable OC2 interrupt
        BCLR    TMSK1,X Bit6


end_OC2ISR
        RTI

************************************************************************
*       Subroutine:     Init_servos                                    *
*       Function:       initializes left and right servo output        *
*                       compares (OC3 and OC4)                         *
```

```
*       Input:          none                                           *
*       Output:         none                                           *
*       Calls:          none                                           *
*       Destroys:       none                                           *
************************************************************************

Init_servos:

       PSHX
       PSHA
       PSHB

       LDX     #BASE

       LDD     #750
       STD     Lduty
       STD     Rduty

* clear OC4 Flag
  BSET    TFLG1,X Bit4
  BSET    TFLG1,X Bit5

  CLI

* enable OC4 interrupt

        BSET    TMSK1,X Bit4
        BSET    CFORC,X Bit4
* enable OC3 interrupt
        BSET    TMSK1,X Bit5
        BSET    CFORC,X Bit5




        PULB
        PULA
        PULX
        RTS


************************************************************************
*       Interrupt Service Routine: OC4ISR                              *
*       Function:       Controls the left servo                        *
*       Input:          Duty cycle                                     *
*       Output:         a specified waveform on PortA pin 4 (OC4)      *
*       Calls:          none                                           *
*       Destroys:       none                                           *
************************************************************************

OC4ISR
        LDX     #BASE
        BRCLR   TFLG1,X Bit4 end_OC4ISR   ;Igonore Illegal Interrupt


         LDAA    #Bit4
```

```
         STAA    TFLG1,X ;clear Flag

         BRSET   PORTA,X Bit4 high
         BSET    TCTL1,X Bit3  ;currently low make high next cycle
         BSET    TCTL1,X Bit2

         LDD     #period
         SUBD    Lduty
         ADDD    TOC4,X
         STD     TOC4,X


        BRA     end_OC4ISR
high:
        BCLR    TCTL1,X Bit2  ;currently high make low next time
        BSET    TCTL1,X Bit3
        LDD     Lduty
        ADDD    TOC4,X
        STD     TOC4,X

end_OC4ISR
        RTI


*************************************************************************
*       Interrupt Service Routine: OC3ISR                               *
*       Function:       Controls the right servo                        *
*       Input:          Duty cycle                                      *
*       Output:         a specified waveform on PortA pin 4 (OC4)       *
*       Calls:          none                                            *
*       Destroys:       none                                            *
*************************************************************************

OC3ISR
       LDX     #BASE
       BRCLR   TFLG1,X Bit5 end_OC3ISR   ;Igonore Illegal Interrupt
       LDAA    #Bit5
       STAA    TFLG1,X ;clear Flag

       BRSET   PORTA,X Bit5 high2
       BSET    TCTL1,X Bit5  ;currently low make high next cycle
       BSET    TCTL1,X Bit4

       LDD     #period
       SUBD    Rduty
       ADDD    TOC3,X
       STD     TOC3,X


       BRA     end_OC3ISR

high2:
        BCLR    TCTL1,X Bit4  ;currently high make low next time
        BSET    TCTL1,X Bit5
        LDD     Rduty
        ADDD    TOC3,X
        STD     TOC3,X
```

```
end_OC3ISR
         RTI

************************************************************************
*       Subroutine:     Init_TIC1                                      *
*       Function:       initializes Timer Input Capture1 (TIC1)        *
*       Input:          none                                           *
*       Output:         none                                           *
*       Calls:          none                                           *
*       Destroys:       none                                           *
************************************************************************

Init_TIC1:
         PSHX
         LDX     #BASE

* Set INC1 to capture on the rising edge
        BCLR    TCTL2,X Bit5
        BSET    TCTL2,X Bit4

* clear INC1 Flag
        BSET    TFLG1,X Bit2
* enable INC1 innterupt
        BSET    TMSK1,X Bit2

        CLI
        PULX
        RTS


************************************************************************
*       Interrupt Service Routine: IC1_ISR                             *
*       Function:       sample input pulses                            *
*                       (used to read heading from the compass)        *
*       Input:          none                                           *
*       Output:         heading                                        *
*       Calls:          none                                           *
*       Destroys:       none                                           *
************************************************************************

IC1_ISR:
         LDX     #BASE   ;3 cycles

         BRCLR   TFLG1,X Bit2 end_IC1 ;ignore invalid interrupt
         BSET    TFLG1,X Bit2         ;clear the flag
         LDD     TIC1,X               ; read the time of interrupt

         BRSET   TCTL2,X Bit5  fall   ;branch if reading falling edge
         STD     rising_edge

**      set to capure falling edge
        BCLR    TCTL2,X Bit4
        BSET    TCTL2,X Bit5
        BRA     end_IC1
```

```
fall    SUBD    rising_edge
        LSLD
        SUBA    #04
        STAA    heading



**      set to capture rising edge
       BCLR    TCTL2,X Bit5
       BSET    TCTL2,X Bit4

end_IC1 RTI
************************************************************************
*       Subroutine:     Read_IR                                        *
*       Function:       reads the values of  the A/D port and stores   *
*                       then to L_IR, R_IR, LF_IR, RF_IR               *
*       Input:          none                                           *
*       Output:         none                                           *
*       Calls:          none                                           *
*       Destroys:       none                                           *
***********************************************************************

Read_IR:
        PSHX
        PSHY
        PSHA
        PSHB
        LDX #BASE
        LDY #ADR1

        BSET    OPTION,X Bit7  ;enable A/D system
        BSET    ADCTL,X Bit4 ; set for multiple scan
        BCLR    ADCTL,X Bit2   ; read Analog 0 - 3
        BCLR    ADCTL,X Bit3   ;
        BCLR    ADCTL,X Bit5   ;single scan

RA1     BRCLR   ADCTL,X Bit7 RA1  ; wait util conversion is complete
        LDAA    0,Y
        STAA    LF_IR
        LDAA    1,Y
        STAA    RF_IR
        LDAA    2,Y
        STAA    R_IR
        LDAA    3,Y
        STAA    L_IR

        PULB
        PULA
        PULY
        PULX
        RTS

************************************************************************
*       Subroutine:     Read_Bump                                      *
*       Function:       reads the values of  the A/D port and stores   *
```

```
*                       then to R_Bump, L_Bump, B_Bump, F_Bump        *
*       Input:          none                                          *
*       Output:         none                                          *
*       Calls:          none                                          *
*       Destroys:       none                                          *
**********************************************************************

Read_Bump:
        PSHX
        PSHY
        PSHA
        PSHB
        LDX #BASE
        LDY #ADR1

        BSET    OPTION,X Bit7  ;enable A/D system
        BSET    ADCTL,X Bit4 ; set for multiple scan
        BSET    ADCTL,X Bit2   ; read Analog 0 - 3
        BCLR    ADCTL,X Bit3   ;
        BCLR    ADCTL,X Bit5   ;single scan

RB1     BRCLR   ADCTL,X Bit7 RB1  ; wait util conversion is complete
        LDAA    0,Y
        STAA    R_Bump
        LDAA    1,Y
        STAA    L_Bump
        LDAA    2,Y
        STAA    F_Bump
        LDAA    3,Y
        STAA    B_Bump

        PULB
        PULA
        PULY
        PULX
        RTS


***********************************************************************
*       Subroutine:     turn_left                                     *
*       Function:       turns left the number of degreess stored in   *
*                       amount (unsigned value)                       *
*       Input:          degrees, curent heading                       *
*       Output:         turns robot left                              *
*       Calls:          Read_Bump                                     *
*       Destroys:       none                                          *
**********************************************************************

turn_left
        PSHA
        PSHB



        LDAA    #FALSE
        STAA    error_flag
```

```
        LDAA    heading
        CMPA    degrees
        BLO     TL1     ; if the number of degrees is smaler than the
*                       ; heading
        SUBA    degrees
        STAA    temp1


TL2
        STAA    temp2
        LDD     #STOP
        STD     Lduty
        LDD     #RightForward
        STD     Rduty
        JSR     Read_Bump


        LDAA    #10
        STAA    temp2
TL5     LDAA    #10  ;100 ms delay
        STAA    delayTime2
        JSR     delay2

TL4     JSR     Read_Bump
        LDAA    #150
        CMPA    L_Bump
        BHI     TL3     ;the robot hit something

        LDAA    #TRUE
        CMPA    T_flag
        BNE     TL4
        DEC     temp2
        BNE     TL5


        LDD     #STOP
        STD     Rduty


        LDAA    #$FF
        STAA    temp2
TL6     DEC     temp2
        BNE     TL6



        LDAA    heading
**      ANDA    #$F0
        STAA    temp2
**      JSR     OutByte
        LDAA    temp1
**      ANDA    #$F0
        CMPA    temp2
        BNE     TL2


        BRA     end_turn_left
```

```
TL1     LDAA    #Max_heading   ; max compass heading
        SUBA    degrees
        ADDA    heading
        STAA    temp1
        BRA     TL2

TL3     LDAA    #TRUE   ;set the error flag if the robot hit something
        STAA    error_flag

end_turn_left:
        LDD     #STOP
        STD     Lduty
        STD     Rduty
        PULB
        PULA
        RTS

************************************************************************
*       Subroutine:     turn_right                                     *
*       Function:       turns right the number of degreess stored in   *
*                       amount (unsigned value)                        *
*       Input:          degrees, curent heading                        *
*       Output:         turns robot right                              *
*       Calls:          Read_Bump                                      *
*       Destroys:       none                                           *
***********************************************************************

turn_right
        PSHA
        PSHB

        LDAA    #FALSE
        STAA    error_flag
        LDAA    heading
        ADDA    degrees
        CMPA    #Max_heading
        BHI     TR1     ; if the new heading is larger than the
*                       ; max heading

        STAA    temp1

TR2     LDD     #STOP
        STD     Rduty
        LDD     #LeftForward
        STD     Lduty

        LDAA    #$FF
        STAA    temp2
        LDAA    #10
        STAA    temp2

TR5     LDAA    #10  ;10 ms delay
        STAA    delayTime2
        JSR     delay2

TR4
```

```
        JSR     Read_Bump
        LDAA    #150
        CMPA    R_Bump
        BHI     TR3     ;the robot hit something
        LDAA    #TRUE
        CMPA    T_flag
        BNE     TR4
        DEC     temp2
        BNE     TR5

        LDAA    #100 ;10 ms delay
        STAA    delayTime2
        JSR     delay2

        LDD     #STOP
        STD     Lduty

        LDAA    #TRUE
TR6     CMPA    T_flag
        BNE     TR6

        LDAA    heading
**      ANDA    #$F0
        STAA    temp2
        LDAA    temp1
**      ANDA    #$F0
        CMPA    temp2
        BNE     TR2
        BRA     end_turn_right

TR1     SUBA    #Max_heading
        STAA    temp1
        BRA     TR2

TR3     LDAA    #TRUE   ;set the error flag if the robot hit something
        STAA    error_flag

end_turn_right:
        LDD     #STOP
        STD     Lduty
        STD     Rduty
        PULB
        PULA
        RTS

**********************************************************************
*       SUBROUTINE      -  Drive
*  Description this subroutine will case the robot to drive around
* if the robot sees an obstacle in front of it it will look left and
* turn left if if can if not it will look right and then turn right
* if it cannot turn left of right it will back up
**********************************************************************

Drive:
        LDD     #LeftForward
        STD     Lduty
        LDD     #RightForward
```

```
        STD     Rduty
look    JSR     Read_IR
        JSR     Read_Bump
**      check front sensors
        LDAA    #100
        CMPA    LF_IR
        BLO     backup
        CMPA    RF_IR
        BLO     backup

        LDAA    #32
        CMPA    LF_IR
        BLO     turn
        CMPA    RF_IR
        BLO     turn
        CMPA    F_Bump
        BHI     backup
        BRA     Drive

turn    LDAA    #100
        CMPA    L_IR
        BHI     left
        CMPA    R_IR
        BHI     right

backup  LDD     #LeftReverse
        STD     Lduty
        LDD     #RightReverse
        STD     Rduty
        BRA     look

left    LDD     #STOP
        STD     Lduty
        LDD     #RightForward
        STD     Rduty
        LDAA    #64
        CMPA    L_Bump
        BHI     right
        BRA     look

right   LDD     #STOP
        STD     Rduty
        LDD     #LeftForward
        STD     Lduty
        LDAA    #64
        CMPA    R_Bump
        BHI     left
        BRA     look


**********************************************************************
*       SUBROUTINE      -  Drive2
*  Description this subroutine will case the robot to drive around
* if the robot sees an obstacle in front of it it will look left and
* turn left if if can if not it will look right and then turn right
* if it cannot turn left of right it will back up
* This subroutine will use the compass to make 90 degree turns
```

```
**********************************************************************

Drive2:
        LDAA    #$AA
        JSR     OutByte

        LDD     #LeftForward
        STD     Lduty
        LDD     #RightForward
        STD     Rduty
look2   JSR     Read_IR
        JSR     Read_Bump
**      check front sensors
        LDAA    #100
        CMPA    LF_IR
        BLO     backup2
        CMPA    RF_IR
        BLO     backup2

        LDAA    #75
        CMPA    LF_IR
        BLO     turnR

        CMPA    RF_IR
        BLO     turnL

        CMPA    F_Bump
        BHI     backup2
        BRA     Drive2

turnL
        LDAA    #100
        CMPA    L_IR
        BHI     backup2
        BRA     left2

turnR   LDAA    #100
        CMPA    R_IR
        BHI     backup2
        BRA     right2

backup2
        LDD     #LeftReverse
        STD     Lduty
        LDD     #RightReverse
        STD     Rduty
        BRA     look2


left2   LDAA    #35
        STAA    degrees
        JSR     turn_left
**        LDAA    error_flag
**        BEQ     right2
        BRA     look2

right2  LDAA    #35
```

```
        STAA    degrees
        JSR     turn_right
**        LDAA    error_flag
**        BEQ     left2
        BRA     look2


************************************************************************
*       SUBROUTINE      -  Drive3                                      *
* Description this subroutine will case the robot to drive strait for  *
* a set time and then turn either left or right the specifiecd amount  *
*       Input   :       distance
*                       degrees
*                       direction
************************************************************************

Drive3:
**        LDAA    #$AA
**        JSR     OutByte


d3a     LDAA    #10
        STAA    CNT

        LDD     #LeftForward
        STD     Lduty
        LDD     #RightForward
        STD     Rduty

d3b     LDAA    #100
        STAA    delayTime2
        JSR     delay2

look3   JSR     Read_IR
        JSR     Read_Bump
**      check front sensors
        LDAA    #100
        CMPA    LF_IR
        BLO     stop
        CMPA    RF_IR
        BLO     stop

        LDAA    #TRUE
        CMPA    T_flag
        BNE     look3
        DEC     CNT
        BNE     d3b
        DEC     distance
        BNE     d3a

        LDAA    direction
        CMPA    #Right

        BEQ     d3c
        JSR     turn_left
        BRA     end_drive3
```

```
d3c     JSR     turn_right
        BRA     end_drive3

stop    LDD     #STOP
        STD     Lduty
        STD     Rduty
        LDAA    TRUE
        STAA    error_flag
        BRA     end_drive3

end_drive3
        RTS


************************************************************************
**
*       Subroutine:     Shapes
*
*       Description:   makes a series of shapes that are stored in the
*
*                       shapes table
*
*       Input:          starting value of shape
************************************************************************
**

Shapes:
        LDAA    0,X
        STAA    distance
        JSR     OutByte
        INX
        LDAA    0,X
        STAA    degrees
        JSR     OutByte
        INX
        LDAA    0,X
        STAA    direction
        JSR     OutByte

        JSR     Drive3
        LDAA    #TRUE

        CMPA    #error_flag
        BEQ     end_shapes

        INX
        LDAA    0,X
        CMPA    #EOS
        BNE     Shapes

end_shapes:

        RTS

***********************************************************************

************************************************************************
```

```
* Input          : none
***********************************************************************

test2:
        PSHA
t2a     LDX     #SetSpeed
        JSR     OutStr
        LDX     #Lduty
        LDAA    0,X
        JSR     OutByte
        LDAA    1,X
        JSR     OutByte
        LDAA    #CR
        JSR     OutChar
        LDAA    #LF
        JSR     OutChar

        LDX     #Rduty
        LDAA    0,X
        JSR     OutByte
        LDAA    1,X
        JSR     OutByte

        JSR     InChar
        CMPA    #$31
        BEQ     incSpeed
        CMPA    #$32
        BEQ     decSpeed
        CMPA    #$33
        BEQ     incSpeed2
        CMPA    #$34
        BEQ     decSpeed2

        BRA     end_test2

incSpeed
         LDD     Lduty
         ADDD    #150
         STD     Lduty
         BRA     t2a

decSpeed
         LDD     Lduty
         SUBD    #150
         STD     Lduty
         BRA     t2a

incSpeed2
         LDD     Rduty
         ADDD    #300
         STD     Rduty
         BRA     t2a

decSpeed2
```

```
        LDD     Rduty
        SUBD    #300
        STD     Rduty
        BRA     t2a

end_test2
        PULA
        RTS

*************************************************************************
*                       SUBROUTINE - test3
* Description   : this routine will test the compass of the robot
left turns intersparced by periods of srait li
* Input         : none
*************************************************************************
test3
       PSHA
       LDX     #CompassHeading
       JSR     OutStr
       JSR     InChar
       LDAA     heading
       JSR     OutByte
       PULA
       RTS

*************************************************************************
*                       SUBROUTINE - test4
* Description   : this routine will test delay routine of the robot

* Input         : none
*************************************************************************
test4
       PSHA
       LDAA     #$2A
       JSR      OutChar
t4b    LDAA     #100
       STA      delayTime2
       JSR      delay2
       LDAA     #TRUE
t4a    CMPA     T_flag
       BNE      t4a
       LDAA     #$2A
       JSR      OutChar
       BRA      t4b
end_test4
       PULA
       RTS

*************************************************************************
*                       SUBROUTINE - test5
* Description   : this routine will test the IR
* Input         : none
*************************************************************************
test5
       PSHA
       PSHX
```

```
        JSR     Read_IR
        LDX     #IR1
        JSR     OutStr
        LDAA    L_IR
        JSR     OutByte

        LDX     #IR2
        JSR     OutStr
        LDAA    R_IR
        JSR     OutByte

        LDX     #IR3
        JSR     OutStr
        LDAA    LF_IR
        JSR     OutByte

        LDX     #IR4
        JSR     OutStr
        LDAA    RF_IR
        JSR     OutByte

        PULX
        PULA
        RTS

*************************************************************************
*                       SUBROUTINE - test6
* Description   : this routine will test the Bump sensors on the robot
* Input         : none
*************************************************************************
test6
        PSHA
        PSHX

        JSR     Read_Bump
        LDX     #B1
        JSR     OutStr
        LDAA    L_Bump
        JSR     OutByte

        LDX     #B2
        JSR     OutStr
        LDAA    R_Bump
        JSR     OutByte

        LDX     #B3
        JSR     OutStr
        LDAA    F_Bump
        JSR     OutByte

        LDX     #B4
        JSR     OutStr
        LDAA    B_Bump
        JSR     OutByte

        PULX
        PULA
```

```
        RTS


*************************************************************************
*                        END OF PROGRAM
*************************************************************************
```

Research about mouse data

PS/2 Mouse/Keyboard Protocol
Copyright 1999 Adam Chapweske

NOTE: THIS SERVER IS A LITTLE FLAKY... IF ANY IMAGES DO NOT LOAD, CLICK "RELOAD" ON
YOUR BROWSER A FEW TIMES AND THE PICTURES WILL EVENTUALLY APPEAR.

Intruduction:

The PS/2 device interface, used by many modern mice and keyboards, was developed by IBM and originally appeared in
the IBM Technical Reference Manual. However, this manual has not been printed for many years and as far as I know,
there is currently no official publication of this information. I have not had access to the IBM Technical Reference
Manual, so all information on this page comes from my own experiences with the mouse and keyboard, as well as help
from the references listed at the bottom of this page.

This document descibes the interface used by the PS/2 mouse and AT (PS/2) keyboard. I'll cover the physical and
electrical interface, as well as the protocol. If you need higher-level information, such as commands, data packet formats,
or other information specific to the keyboard or mouse, I have written separate documents for the two devices:

The AT Keyboard Interface (same as PS/2 keyboard)
The PS/2 Mouse Interface

I also encourage you to check out my homepage for more information related to this topic, including projects, code, and
links related to the mouse and keyboard.

The Connector:

The physical keyboard/mouse port is one of two styles of connectors: The 5-pin DIN or the 6-pin mini-DIN. Both
connectors are completely (electrically) similar; the only practical difference between the two is the arrangement of pins.

This means that the two types of connectors can easily be changed with simple hard-wired adaptors. These cost about
$6 each or you can make your own by matching the pins on any two connectors. The DIN standard was created by the
German Standardization Organization (Deutsches Institut fuer Norm) . Their website is at http://www.din.de (this site is in
German, but most of their pages are also available in English.)

PC keyboards can have either a 6-pin mini-DIN or a 5-pin DIN connector. If your keyboard has a 6-pin mini-DIN and
your computer has a 5-pin DIN (or visa versa), the two can be made compatible with the adaptors described above.
Keyboards with the 6-pin mini-DIN are often referred to as "PS/2" keyboards, while those with the 5-pin DIN are called
"AT" or "XT" devices. XT keyboards are quite old and haven't been made for about ten years. All modern keyboards
built for the PC are either PS/2, AT, or USB. This document does not apply to USB devices, which use a completely
different interface.

Mice come in a number of shapes and sizes (and interfaces.) The most popular type is probably the PS/2 mouse, with
USB mice slowly gaining popularity. Serial mice are also quite popular, but the computer industry is abandoning them in
support of USB and PS/2 devices. This document applies only to PS/2 mice. If you want to interface a serial mouse,
check out Microchip's appnote #519, "Implementing a Simple Serial Mouse Controller."

As a side note, there is one other type of connector you may run into on keyboards. While most keyboard cables are
hard-wired to the keyboard, there are some whose cable is not permanently attached and come as a separate
component. These cables have a DIN connector on one end (the end that connects to the computer) and a SDL
(Sheilded Data Link) connector on the keyboard end. SDL was created by a company called "AMP." This connector is
somewhat similar to a telephone connector in that it has wires and springs rather than pins, and a clip holds it in place. If
you need more information on this connector, you might be able to find it on AMP's website at
http://www.connect.amp.com. I have only seen this type of connector on (old) XT keyboards, although there may be AT
keyboards that also use the SDL. Don't confuse the SDL connector with the USB connector--they probably both look
similar in my diagram below, but they are actually very different. Keep in mind that the SDL connector has springs and

moving parts, while the USB connector does not.

The pinouts for each connector are shown below:
(If any of these images do not load, hit "reload" on your browser a few times.)

Male

(Plug)

Female

(Socket)

5-pin DIN (AT/XT):
1 - Clock
2 - Data
3 - Not Implemented
4 - Ground
5 - +5v

Male

(Plug)

Female

(Socket)

6-pin Mini-DIN (PS/2):
1 - Data
2 - Not Implemented
3 - Ground
4 - +5v
5 - Clock
6 - Not Implemented

6-pin SDL:
A - Not Implemented
B - Data
C - Ground
D - Clock
E - +5v
F - Not Implemented

General Description:

(Note: Throughout this document, I may use the more general term "host" to refer to the computer--or whatever the
keyboard/mouse is connected to-- and the term "device" will refer to the keyboard/mouse.)

There are four interesting pins on the connectors just described: Ground, +5v, Data, and Clock. The +5V is supplied
by the host (computer) and the keyboard/mouse's ground is connected to the host's electrical ground. Data and Clock
are both open collector, which means they are normally held at a high logic level but can easily be pulled down to ground
(logic 0.) Any device you connect to a PS/2 mouse, keyboard, or host should have large pull-up resistors on the Clock
and Data lines. You apply a "0" by pulling the line low and you apply a "1" by letting the line float high. Refer to Figure 1
for a general interface to Data and Clock. (Note: if you are going to use a microcontroller such as the PIC, where I/O is
bidirectional, you may skip the transistors and buffers and use the same pin for both input and output. With this
configuration, a "1" is asserted by setting the pin to input and let the resistor pull the line high. A "0" is then asserted by
changing the pin to output and write a "0" to that pin, which will pull the line to ground.)

Figure 1: Open-collector interface to Data and Clock. Data and Clock are read on the microcontroller's port A and
B, respectively. Both lines are normally held at +5V, but can be pulled to ground by asserting logic 1 on C and D. As a
result, Data equals D, inverted, and Clock equals C, inverted.

The PS/2 mouse and keyboard implement a bidirectional synchronous serial protocol. In other words, Data is sent one
bit at a time on the Data line and is read on each time Clock is pulsed. The keyboard/mouse can send data to the host
and the host can send data to the device, but the host always has priority over the bus and can inhibit communication from
the keyboard/mouse at any time by holding Clock low.

Data sent from the keyboard/mouse to the host is read on the falling edge of the clock signal (when Clock goes from high to low); data sent from the host to the keyboard/mouse is read on the rising edge (when Clock goes from low to high.) Regardless of the direction of communication, the keyboard/mouse always generates the clock signal. If the host wants to send data, it must first tell the device to start generating a clock signal (that process is described in the next section.) The maximum clock frequency is 33 kHz and most devices operate within 10-20kHz. If you want to build a PS/2 device, I would recommend keeping this frequency around 15 kHz. This means Clock should be high for about 40 microseconds and low for 40 microseconds.

All data is arranged in bytes with each byte sent in a frame consisting of 11-12 bits. These bits are:

1 start bit. This is always 0.
8 data bits, least significant bit first.
1 parity bit (odd parity).
1 stop bit. This is always 1.
1 acknowledge bit (Host-to-device communication only)

The parity bit is set if there is an even number of 1's in the data bits and reset (0) if there is an odd number of 1's in the data bits. The number of 1's in the data bits plus the parity bit always add up to an odd number (odd parity.) This is used for error detection.

When the host is sending data to the keyboard/mouse, a handshaking bit is sent from the device to acknowledge the packet was received. This bit is not present when the device sends data to the host.

Device-to-Host Communication:

The Data and Clock lines are both open collector (normally held at a high logic level.) When the keyboard or mouse wants to send information, it first checks Clock to make sure it's at a high logic level. If it's not, the host is inhibiting communication and the device must buffer any to-be-sent data until it regains control of the bus (the keyboard has a 16-byte buffer and the mouse's buffer stores only the last packet sent.) If the Clock line is high, the device can begin to transmit its data.

As I mentioned in the previous section, the keyboard and mouse use a serial protocol consisting of 11-bit frames. These
bits are:

1 start bit. This is always 0.
8 data bits, least significant bit first.
1 parity bit (odd parity).
1 stop bit. This is always 1.

Each bit is read by the host on the falling edge of the clock, as is illustrated in Figures 2 & 3.

Figure 2: Device-to-host communication. The Data line changes state when Clock is high and that data is latched on the
falling edge of the clock signal.

Figure 3: Scan code for the "Q" key (15h) being sent from a keyboard to the computer. Channel A is the Clock signal;
channel B is the Data signal.

The clock frequency is 10-16.7kHz. The time from the rising edge of a clock pulse to a Data transition should be at least
5 microseconds. The time from a data transition to the falling edge of a clock pulse should be at least 5 microseconds and
no greater than 25 microseconds. This timing is very important--you should follow it exactly. The host may pull the line
low before the 11th clock pulse (stop bit), causing the device to abort sending the current byte (this is very rare.) After
the stop bit is transmitted, the device should wait at least 50 microseconds before sending the next packet. This gives the
host time to inhibit transmission while it processes the received byte (the host will usually automatically do this after each
packet is received.) The device should wait at least 50 microseconds after the host releases an inhibit before sending any
data.

I would recommend the following process for sending a single byte from an emulated keyboard/mouse to the host:

1) Wait for Clock = high.

2) Delay 50 microseconds.
3) Clock still = high?
   No--goto step 1
4) Data = high?
   No--Abort (and read byte from host)
5) Delay 20 microseconds (=40 microseconds to the time Clock is pulled low in sending the start bit.)
6) Output Start bit (0) \ After sending each of these bits, test
7) Output 8 data bits > Clock to make sure host hasn't pulled it
8) Output Parity bit / low (which would abort this transmission.)
9) Output Stop bit (1)
10) Delay 30 microseconds (=50 microseconds from the time Clock is released in sending the stop bit)

The process for sending a single bit should then be as follows:

1) Set/Reset Data
2) Delay 20 microseconds
3) Bring Clock low
4) Delay 40 microseconds
5) Release Clock
6) Delay 20 microseconds

Here is some sample code written for the PIC16F84 that follows the above algorithms to send a byte to the host.
"Delay" is a self-explanitory macro; "CLOCK" and "DATA" are the bits connected to the Clock and Data lines; "TEMP0",
"PARITY", and "COUNTER" are all general purpose registers. Note that in the "PS2outBit" routine, the Data and Clock
lines are brought low by setting the appropriate I/O pin to output (it's assumed their output was set to "0" at the beginning
of the program.) And they are allowed to float (high) by setting the I/O pin to input (and allow a pull-up resistor to pull
the line high.) This was written for a PIC running at 4.61 MHz +/- 25% (RC oscillator: 5k/20pF). This is very important
for timing considerations.

```
ByteOut       movwf   TEMP0          ;Save to-be-sent byte
InhibitLoop   btfss   CLOCK          ;Check for inhibit
              goto    InhibitLoop
              Delay   50             ;Delay 50 microseconds
              btfss   CLOCK          ;Check again for inhibit
              goto    InhibitLoop
              btfss   DATA           ;Check for request-to-send
              retlw   0xFF
              clrf    PARITY         ;Init reg for parity calc
              movlw   0x08
```

```
            movwf   COUNTER
            movlw   0x00
            call    BitOut        ;Output Start bit (0)
            btfss   CLOCK         ;Test for inhibit
            goto    ByteOutEnd
            Delay   4
ByteOutLoop     movf    TEMP0, w
            xorwf   PARITY, f     ;Calculate parity
            call    BitOut        ;Output Data bits
            btfss   CLOCK         ;Test for inhibit
            goto    ByteOutEnd
            rrf     TEMP0, f
            decfsz  COUNTER, f
            goto    ByteOutLoop
            Delay   2
            comf    PARITY, w
            call    BitOut        ;Output Parity bit
            btfss   CLOCK         ;Test for inhibit
            goto    ByteOutEnd
            Delay   5
            movlw   0xFF
            call    BitOut        ;Output Stop bit (1)
            Delay   48
            retlw   0x00

ByteOutEnd      bsf     STATUS, RP0     ;Host has aborted
            bsf     DATA          ;DATA=1
            bsf     CLOCK         ;CLOCK=1
            bcf     STATUS, RP0
            retlw   0xFE

BitOut          bsf     STATUS, RP0
            andlw   0x01
            btfss   STATUS, Z
            bsf     DATA
            btfsc   STATUS, Z
            bcf     DATA
            Delay   21
            bcf     CLOCK
            Delay   45
            bsf     CLOCK
            bcf     STATUS, RP0
            Delay   5
            return
```

Host to Device Communication:

The packet is sent a little differently in host-to-device communication...

First of all, the PS/2 device always generates the clock signal. If the host wants to send data, it must first put the Clock
and Data lines in a "Request-to-send" state as follows:

Inhibit communication by pulling Clock low for at least 100 microseconds.
Apply "Request-to-send" by pulling Data low, then release Clock.

The device should check for this state at intervals not to exceed 10 milliseconds. When the device detects this state, it will
begin generating Clock signals and clock in eight data bits and one stop bit. The host changes the Data line only when
the Clock line is low, and data is latched on the rising edge of the clock pulse. This is opposite of what occours in
device-to-host communication.

After the stop bit is sent, the device will acknowledge the received byte by bringing the Data line low and generating one
last clock pulse. If the host does not release the Data line after the 11th clock pulse, the device will continue to generate
clock pulses until the the Data line is released (the device will then generate an error.)

The Host may abort transmission at time before the 11th clock pulse (acknowledge bit) by holding Clock low for at least
100 microseconds.

To make this process a little easier to understand, here's the steps the host must follow to send data to a PS/2 device:

1) Bring the Clock line low for at least 100 microseconds.
2) Bring the Data line low.
3) Release the Clock line.
4) Wait for the device to bring the Clock line low.
5) Set/reset the Data line to send the first data bit
6) Wait for the device to bring Clock high.
7) Wait for the device to bring Clock low.
8) Repeat steps 5-7 for the other seven data bits and the parity bit
9) Release the Data line.
10) Wait for the device to bring Data low.
11) Wait for the device to bring Clock low.
12) Wait for the device to release Data and Clock

Figure 3 shows this graphically and Figure 4 separates the timing to show which signals are generated by the host, and which are generated by the PS/2 device. Notice the change in timing for the Ack bit--the data transition occours when the Clock line is high (rather than when it is low as is the case for the other 11 bits.)

Figure 3: Host-to-Device Communication.

Figure 4: Detailed host-to-device communication.

Figure 4 shows two important timing considerations: (a), and (b). (a), the time it takes the device to begin generating clock pulses after the host initially takes the Clock line low, must be no greater than 15ms; (b), the time it takes for the packet to be sent, must be no greater than 2ms. If either of these time limits is not met, the host will generate an error.
Immediately after the packet is received, the host may bring the Clock line low to inhibit communication while it processes data. If the command sent by the host requires a response, that response must be received no later than 20ms after the host releases the Clock line. If this does not happen, the host generates an error. As was the case with Device-to-host communication, no Data transition may occur with 5 microseconds of a Clock transition.

If you want to emulate a mouse or keyboard, I would recommend reading data from the host as follows:

In your main program, check for Data=low at least every 10 milliseconds.
If Data has been brought low by the host, read one byte from the host

1) Wait for Clock=high
2) Is Data still low?
   No--An error occurred; Abort.
3) Read 8 data bits \ After reading each of these bits, test
4) Read parity bit > Clock to make sure host hasn't pulled it
5) Read stop bit / low (which would abort this transmission.)
6) Data still equals 0?
   Yes--Keep clocking until Data=1 then generate an error
7) Output Acknowledge bit
8) Check Parity bit.
   Generate an error if parity bit is incorrect
9) Delay 45 microseconds (to give host time to inhibit next transmission.)

Read each bit (8 data bits, parity bit, and stop bit) as follows:
1) Delay 20 microseconds
2) Bring Clock low
3) Delay 40 microseconds
4) Release Clock
5) Delay 20 microsecond
6) Read Data line
Send the acknowledge bit as follows:
1) Delay 15 microseconds
2) Bring Data low
3) Delay 5 microseconds
4) Bring Clock low
5) Delay 40 microseconds
6) Release Clock
7) Delay 5 microseconds
8) Release Data

Here is some sample code written for the PIC16F84 that implements the above algorithms to read data from a PS/2 host. "Delay" is a self-explanitory macro; "CLOCK" and "DATA" are the port bits connected to the Clock and Data lines; "TEMP0", "PARITY", and "COUNTER" are all general purpose registers. Note that in the "PS2inBit" routine, Clock is brought low by setting the appropriate I/O pin to output (it's assumed they were set to "0" at the beginning of the program.) And it is allowed to float (high) by setting the I/O pin to input (and allow a pull-up resistor to pull the line high.) Timing was worked out for a PIC running at 4.61 MHz +/- 25% (RC oscillator with values 5k/20 pF). Will work for any oscillator between 3.50 MHz - 5.76 MHz.

```
ByteIn      btfss   CLOCK        ;Wait for start bit
            goto    ByteIn
            btfsc   DATA
            goto    ByteIn
            movlw   0x08
            movwf   COUNTER
            clrf    PARITY       ;Init reg for parity calc
            Delay   28
ByteInLoop  call    BitIn        ;Clock in Data bits
            btfss   CLOCK        ;Test for inhibit
            retlw   0xFE
            bcf     STATUS, C
            rrf     RECEIVE, f
            iorwf   RECEIVE, f
            xorwf   PARITY,f
```

```
            decfsz  COUNTER, f
            goto    ByteInLoop
            Delay   1
            call    BitIn          ;Clock in Parity bit
            btfss   CLOCK          ;Test for inhibit
            retlw   0xFE
            xorwf   PARITY, f
            Delay   5
ByteInLoop1     Delay   1
            call    BitIn          ;Clock in Stop bit
            btfss   CLOCK          ;Test for inhibit
            retlw   0xFE
            xorlw   0x00
            btfsc   STATUS, Z      ;Stop bit = 1?
            clrf    PARITY          No--cause an error condition.
            btfsc   STATUS, Z      ;Stop bit = 1?
            goto    ByteInLoop1     ;  No--keep clocking.

            bsf     STATUS, RP0     ;Acknowledge
            bcf     DATA
            Delay   11
            bcf     CLOCK
            Delay   45
            bsf     CLOCK
            Delay   7
            bsf     DATA
            bcf     STATUS, RP0

            btfss   PARITY, 7      ;Parity correct?
            retlw   0xFF           ;  No--return error

            Delay   45
            retlw   0x00

BitIn           Delay   8
            bsf     STATUS, RP0
            bcf     CLOCK
            Delay   45
            bsf     CLOCK
            bcf     STATUS, RP0
            Delay   21
            btfsc   DATA
            retlw   0x80
            retlw   0x00
```

Other Sources / References:

Adam's micro-Resources Home - Many pages/links to related information.
The AT Keyboard - My page on AT keyboards
The PS/2 Mouse - My page on the PS/2 mouse
Synaptics Touchpad Interfacing Guide -Very informative!
PS/2 Keyboard and Mouse Protocols - Timing diagrams.
Holtek - Informative datasheets on many different PS/2 mice (and other peripherals).

Interfacing the AT Keyboard
Copyright 2001 Adam Chapweske

This document is under construction... I'll post more information as I have time... Click here for the old version of this guide.

Note: This document refers to AT and PS/2 keyboards. The two keyboards are exactly the same except for their connectors. The AT keyboard uses a 5-pin DIN connector, while the PS/2 keyboard uses the 6-pin mini-DIN. That is the only difference.

General Description:

Keyboards consist of a large matrix of keys, all of which are monitored by an on-board processor. The specific processor(1) varies from keyboard-to-keyboard but they all basically do the same thing: Monitor which key(s) are being pressed/released and send the appropriate data to the host. This processor takes care of all the debouncing and buffers any data in its 16-byte buffer, if needed. Your motherboard contains a keyboard controller that is in charge of decoding all of the data received from the keyboard and informing your software of what's going on. All communication between the host and the keyboard uses an IBM protocol.

Electrical Interface / Protocol:

The keyboard uses the same protocol as the PS/2 mouse. Click here for detailed information about that protocol.

Scan Codes:

Your keyboard's processor spends most of its time scanning, or monitoring, the matrix of keys. If it finds that any key is being pressed, released, or held down, the keyboard will send a packet of information known as a scan code to your computer. There are two different types of scan codes: make codes and break codes. A make code is sent when a key is pressed or held down. A break code is sent when a key is released. Every key is assigned its own unique make code and break code so the host can determine exactly what happened to which key simply by looking at a single scan code sent from the keyboard. The set of make and break codes for every key comprises a scan code set. There are three standard scan code sets, named 1, 2, and 3. Scan code set 2 is the default, and is the only set used by all modern PCs. Sets 1 and 3 exist for compatibility with older systems. You may switch scan code sets using the "Set Scan Code Set" (0xF0) command.

So how do you figure out what the scan codes are for each key? Unfortunately, there's no simple formula for calculating this. If you want to know what the make code or break code is for a specific key, you'll have to look it up in a table.
I've composed tables for all make codes and break codes in all three scan code sets:

Scan Code Set 1
Scan Code Set 2
Scan Code Set 3

Make Codes, Break Codes, and Typematic Repeat:

Whenever any key on a keyboard is pressed, that key's make code is sent to the computer. Keep in mind that a make code only represents a key on a keyboard--it does not represent the character printed on that key. This means that there is no defined relationship between a make code and a character. It's up to your software to translate the scan codes to characters or commands. If you want to associate a make code with a character, you'll have to implement a look-up table in your program.

Although most set 2 make codes are only one-byte wide, there are a handfull of extended keys whose make codes are two or four bytes wide. These make codes can be identified by the fact that the first byte is E0h.

Just as a make code is sent to the computer whenever a key is pressed, a break code is sent to the computer whenever a key is released. In addition to every key having its own unique make code, they all have their own unique break code. Fortunately, however, you won't always have to use tables to figure out a key's break code--certain relationships do exist between make codes and break codes. Most set 2 break codes are two bytes long where the first byte is F0h and the second byte is the make code for that key. Break codes for extended keys are usually three bytes long and the first two bytes are E0h, F0h, and the last byte is the last byte of that key's make code. As an example, I have listed below a few set 2 make codes and break codes for some keys:

Key	(Set 2) Make Code	(Set 2) Break Code
"A"	1C	F0,1C
"5"	2E	F0,2E
"F10"	09	F0,09
Right Arrow	E0, 74	E0, F0, 74
Right "Ctrl"	E0, 14	E0, F0, 14

Example: What sequence of make codes and break codes should be sent to your computer for the character "G" to appear in a word processor? Since this is an upper-case letter, the sequence of events that need to take place are: press the "Shift" key, press the "G" key, release the "G" key, release the "Shift" key. The scan codes associated with these events are the following: make code for the "Shift" key (12h), make code for the "G" key (34h), break code for the "G" key(F0h,34h), break code for the "Shift" key (F0h,12h). Therefore, the data sent to your computer would be: 12h, 34h, F0h, 34h, F0h, 12h.

If you press a key, its make code is sent to the computer. When you press and hold down a key, that key becomes typematic, which means the keyboard will keep sending that key's make code until the key is released or another key is pressed. To verify this, open a text editor and hold down the "A" key. When you first press the key, the character "a" immediately appears on your screen. After a short delay, another "a" will appear followed by a whole stream of "a"s until you release the "A" key. There are two important

parameters here: the typematic delay, which is the short delay between the first and second "a", and the typematic rate, which is
how many characters per second will appear on your screen after the typematic delay. The typematic delay can range from 0.25
seconds to 1.00 second and the typematic rate can range from 2.0 cps (characters per second) to 30.0 cps. You may change the
typematic rate and delay using the "Set Typematic Rate/Delay" (0xF3) command.

Command Set:

The following are the only commands that may be sent to the keyboard:

0xFF (Reset) - Keyboard responds with acknowledge (0xFA) then enters Reset mode.
0xFE (Resend) - Keyboard responds by resending the last scan code or command sent to the host.
0xFD (Set Key Type Make) -
0xFC (Set Key Type Make/Break) -
0xFB (Set Key Type Typematic) -
0xFA (Set All Keys Typematic/Make/Break) -
0xF9 (Set All Keys Make) -
0xF8 (Set All Keys Make/Break) -
0xF7 (Set All Keys Typematic) -
0xF6 (Set Default) -
0xF5 (Disable) - Keyboard responds with acknowledge (0xFA), then stops scanning and waits further instructions.
0xF4 (Enable) -
0xF3 (Set Typematic Rate/Delay) - Keyboard responds with acknowledge (0xFA), then waits for the host to send one more
byte, which it also responds to with acknowledge (0xFA). The second byte defines the typematic rate and delay as follows:

Repeat Rate

Bits 0-4
Rate(cps)

Bits 0-4
Rate(cps)

Bits 0-4
Rate(cps)

Bits 0-4
Rate(cps)

00h
2.0

08h
4.0

10h
8.0

18h
16.0

01h
2.1

09h
4.3

11h
8.6

19h
17.1

02h
2.3

0Ah
4.6

12h
9.2

1Ah
18.5

03h
2.5

0Bh
5.0

13h
10.0

1Bh
20.0

04h
2.7

0Ch

			5.5	14h	10.9	1Ch	21.8
05h	3.0	0Dh	6.0	15h	12.0	1Dh	24.0
06h	3.3	0Eh	6.7	16h	13.3	1Eh	26.7
07h	3.7	0Fh	7.5	17h	15.0	1Fh	30.0

Delay

Bits 5-6	Delay (seconds)
00b	

0.25
01b
0.50
10b
0.75
11b
1.00

0xF2 (Read ID) - The keyboard responds with "Acknowledge" (0xFA) follwed by a two-byte device ID of 0x83, 0xAB.

0xF0 (Set Scan Code Set)

0xEE (Echo) - The keyboard responds with "Echo" (0xEE).

0xED (Set/Reset LEDs) -

Initialization:

The following is the communication between my computer and keyboard when it boots-up:

Keyboard: AA Self-test passed
Host: ED Set/Reset Status Indicators
Keyboard: FA Acknowledge
Host: 00 Turn off all LEDs
Keyboard: FA Acknowledge
Host: F2 Read ID
Keyboard: FA Acknowledge
Keyboard: AB First byte of ID
Host: ED Set/Reset Status Indicators
Keyboard: FA Acknowledge
Host: 02 Turn on Num Lock LED
Keyboard: FA Acknowledge
Host: F3 Set Typematic Rate/Delay
Keyboard: FA Acknowledge
Host: 20 500 ms / 30.0 reports/sec
Keyboard: FA Acknowledge
Host: F4 Enable
Keyboard: FA Acknowledge
Host: F3 Set Typematic Rate/delay
Keyboard: FA Acknowledge
Host: 00 250 ms / 30.0 reports/sec
Keyboard: FA Acknowledge

Emulation:

Click here for routines that emulate a PS/2 mouse or keyboard

Footnotes:

1) Some of these processors include:

Holtek: HT82K28A, HT82K628A, HT82K68A, HT82K68E
EMC: EM83050, EM83050H, EM83052H, EM83053H,
Intel: 8048, 8049
Motorola: 6868, 68HC11, 6805
Zilog: Z8602, Z8614, Z8615, Z86C15, Z86E23

Other Sources / References:

Holtek - Informative datasheets on many different AT keyboards (and other peripherals).
PS/2 Mouse/Keyboard Protocol - My page on the protocol used for communication between a keyboard and host.
KB2LCD Keyboard Reader - My keyboard reader with schematics and code.
Scan Codes - My tables of scan codes for various keyboards. Print them out -- they're very handy to have.
Command Sets - My list of commands that can be sent between the host and the keyboard.
Zilog Keyboard Encoder Appnote - Lots of great information on making a keyboard encoder.
Help with keyboard interfacing - Article describing how to interface with AT keyboards.
IBM Keyboard Interfact Project - Good breif article on interfacing to AT and XT keyboards.
PC Keyboard FAQ - Extensive FAQ; large collection of keyboard-related information.
Steve's PC Keyboard info - Links, short FAQ, pinouts, Keyboard viewer software and circuit.
PIC Keyboard Routines - Serial host engine; Keyboard host (8042) emulator; AT Keyboard emulator.
John Voth's Home Page - 8042 Keyboard Controller Schematic.
Philips AN434 - Connecting a PC keyboard to the I2C bus. Examples for the 8XC751 MCU.
AVR AN313 - AT Keyboard-RS232 converter using an AVR MCU. Includes short description/timing diagrams of AT keyboard.

- Adam's micro-Resources -