Lab 2 - Keypad and Display Interface

Introduction

In this lab you will compile and run a C program to read the keypad and write to the 8-character ASCII display on the laboratory computer. You will use these two functions in a demonstration program.

The Laboratory Computer

Each station in the lab is equipped with a PC as well as a microcomputer based on the Motorola 68000 microprocessor (the "lab computer").

The lab computer includes a 'bus' which allows various plug-in printed circuit interface cards to be connected to the microprocessor. These interface cards provide input/output facilities for the laboratory apparatus. In this first lab you will use the keyboard/display card. This card will be used as a control panel and status display in a subsequent experiment.

The laboratory computers run a very simple "operating system" that does not support a compiler. However, it does accept commands to read and alter the contents of the lab computer's memory. Your software will run on the PC and will send commands to the lab computer over a serial interface. These commands will read and write specific memory locations which in turn will control the lab hardware.

The remainder of the laboratory computer system consists of a cabinet and a power supply.

Compiling your Program

Create a project file for your lab using the Turbo C editor. On the first line put the name of your C file (e.g. `LAB2.C`) and on the second line put the file name `IOLIB.LIB`. Save this file with the extension `.PRJ` (e.g. `LAB2.PRJ`).

The project file will allow your program to use the `speek()` and `spoke()` routines which allow the PC to read and write the lab computer's memory by sending commands over the serial port.

Each time you start up Turbo C you will need to define the project name using the Project→Project Name menu item.

If you do not do these steps then Turbo C will not be able to find the `speek()` and `spoke()` functions when you compile your program.

Reading and Writing I/O Ports

The peripherals on the lab computer are controlled by reading and writing values from/to specific locations in the lab computer's memory. These special memory locations are often called "ports" or "registers".

The keypad/display card has a control port and a data port. Each port can be treated as an 8-bit (1 byte) memory location. The addresses (locations in memory) of the control and data ports are:

port	address
data port	0xd0
control port	0xd1

The data port is really two separate ports (an input port and an output port) that appear at the same address.

The value read from an input port corresponds to the logic levels on the data port's input signals. If the input and output signals not connected, then the value read will *not* necessarily be the value on the output port. This is unlike a normal memory location.

The following two functions are available to your Turbo C program to access the lab computer's memory:

```
int speek( int address ) ;
int spoke( int address, int value ) ;
```

The `speek()` function allows you to obtain the value of the memory at a given address and the the `spoke()` function allows you to set the value of a memory location. To use these functions you should

also include the line `#include <iolib.h>` at the start of your program.

Display

The display has 8 character positions. Each position can display an upper-case letter or digit. To display a character on the display, you must write the ASCII value of the character to be displayed or'ed with 0x80 (this sets the most-significant to '1') to the *data* port. Then you must write the following sequence of three values to the *control* port:

- the segment position (the rightmost digit is position zero)
- the segment position or'ed with 0x08
- the segment position

The effect of writing this sequence of values to the control port is to cause a short pulse to appear on bit 3 (value 8) of the control port while the character position remains on the least-significant 3 bits.

For example, to display an 'A' in the leftmost position (position 7), you would write the following sequence of values:

address	value (decimal)	comments
0xd0	65	'A'
0xd1	7	
0xd1	15	7|8
0xd1	7	

Display Handler Function

Write a function that displays the contents of an 8-character array on the lab computer's LED display. The definition of this function will be as follows:

```
void display( char string[] )
{
   ...
}
```

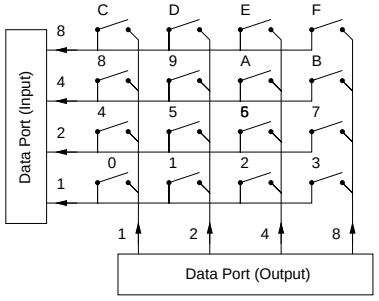
Make sure that your function is declared as shown so that you will be able to re-use it in the next lab.

You may assume the string has at least 8 characters but you should not assume that it is zero-terminated.

Keypad

The keypad consists of 16 push-button switches arranged in a square matrix. Each pushbutton controls a switch which can short (make a connection between) a wire running horizontally and a wire running vertically.

The diagram below is a schematic of the keypad:



The digit or letter above each switch shows the label printed on the switch.

The numbers on the vertical wires show the value that should be written to the output port to "turn on" only that wire.

The numbers on the horizontal wires show the value that will be read from the input port if that wire is "on" (i.e. if the appropriate switch is closed).

The vertical wires are connected to ("driven by") the least significant four bits of the data port output while the horizontal wires are connected to the least significant four bits of the data port input.

Due to the way the keypad matrix has been constructed, a bit on the data port will be read as a '1' only if the switch on that row is pressed *and* a '1' has been written to the data port bit for that switch's column.

The problem is to come up with an algorithm to determine which key (if any) is being pressed. You must do this by writing to the data port to turn on a selected column and then reading the data port to see which (if any) of the rows has a '1' on it.

Note that the upper four bits of the value read from the data port may not be set to zero (they are connected to other switches).

Keypad Handler Function

Write a function `int rdkbd (void)` that waits until a key is pressed and returns a value corresponding to the number pressed (e.g. if the button with the label

'A' is pressed the function should return the value 10).

Before trying to detect whether a button has been pushed your function should wait until all buttons have been released. This will avoid having one button press detected several times.

Demonstration Program

Using the keypad and display handler functions, write a C program that lets the user enter a hexadecimal number from the keypad and displays it on the display unit. The program should do the following:

- initially it should set the display to the last four characters of your student number followed by the first four characters of your surname starting at the leftmost column.
- When a decimal key (0 to 9) is pressed, scroll the display one position to the left and display the new character on the rightmost character position.
- keys between 'A' and 'D' should be ignored
- when the 'E' key is pressed, the display should be reset to the initial number/name display
- when the 'F' key is pressed, the program should terminate

Hints

Begin with an informal ("pseudo-code") description of your program. Don't worry about the details at this stage ("top-down design").

Break down your program into small functions, each of which does something simple that is easy to test. For example, in addition to the two functions given above, you may want to write functions to shift the contents of an array, to wait for all buttons to be released, to set the contents of the array back to their initial value, etc.

Make sure you understand the description of the hardware interface in detail before you begin to write and debug your program. You may want to write one or more short test programs while writing your functions to verify that the hardware indeed operates as you expect ("bottom-up implementation").

Test and debug each part of your program individually ("unit test"). This will make it easier to find bugs.

Use `#define` to define constants such as port addresses or bit masks.

The following code waits for all buttons to be released:

```
spoke ( 0xd0, 0xF ) ;
while ( speek ( 0xd0 ) & 0xF ) { } ;
```

The following code sets the third column of the keypad matrix to a logic '1' and then test whether the top-most row is a logic '1'. If so, then the `E` key has been pressed.

```
spoke ( 0xd0, 0x4 ) ;
if ( speek ( 0xd0 ) & 0x8 ) {
   ...
}
```

You may use the lab at any time since no other groups are using the lab this term. The TA will only be available during the scheduled times.

Get an early start, this lab is more difficult than it looks!

Demonstration

Demonstrate the proper operation of the keypad/display demonstration program to the lab demonstrator. He will ask you some questions to make sure you understand the code you've written.

Write-up

Submit a program listing for the keypad/display demonstration program. It should include comments as described in Lab 1.

Keep the keypad handler and display routines for use in the next lab.