Laboratory Exercise 4
Programming with Interrupts

Goals
After this laboratory exercise, you should have some experience of developing programs in C using interrupts.

Literature
- Brorsson: 5.1–5.2
- Patterson and Hennessy: Chapter 2, 5.7, Appendix A.7, A.10.
- MIPS Lab Environment Reference Manual

Preparation
Read the literature and this laboratory exercise in detail and solve the home assignments.

Home Assignment 1
Study the following C program. It performs the same task as the assembly program of Laboratory Exercise 1, Home Assignment 2. It reads the position of the switches on the lab board and shows this position with the LEDs. Study the code and make sure that you understand how it works.

The C notation char *Switches indicates that the variable Switches is of the type pointer to char, that is, it contains the address of a memory location in which something of type char is stored. In C, the type char is used for ASCII text characters, but it actually works as a small (8-bit) integer. C has a general notation for pointers (addresses). The expression &x means the address of x, and *p means the contents of the memory location that the pointer p points to. When the program is executing, it can be stopped by pressing the reset button on the lab computer.

/ *
* Laboratory Exercise 5, Home Assignment 1
* Written by Jan Eric Larsson, 20 November 1998
* /
#define FALSE 0
#define TRUE 1

unsigned char *Switches = (char*) 0xbf900000;
unsigned char *LEDs     = (char*) 0xbf900000;

void main ()
{
    while (TRUE) *LEDs = *Switches;
}

Home Assignment 2
Study the following C program. It is similar to the program in Home Assignment 1, but only transfers the switch position to the LEDs when an interrupt is activated by the pressing of a button. Note that the file interrupts.s must be added to the project, since it contains the definition of install_normal_int. Note that you must reset the switch information by writing any value to the address 0xbfa00000, as shown in the code.
#include <excepthdr.h>

unsigned char* Switches = (char*) 0xbf900000;
unsigned char* LEDs     = (char*) 0xbf900000;
unsigned char* Reset    = (char*) 0xbfa00000;

void InterruptHandler ()
{
    *LEDs = *Switches;
    *Reset = 0;
}

void main ()
{
    init_ext_int();
    install_normal_int(InterruptHandler);
    enable_int(EXT_INT3);
    while (TRUE) {};
}

Home Assignment 3

Study the following C program stub. It is the skeleton for a program to control the traffic light board. This is a little LED display that simulates a two-way traffic light.

#include <excepthdr.h>

#define MAIN_GRE 0x01
#define MAIN_YEL 0x02
#define MAIN_RED 0x04
#define SIDE_GRE 0x10
#define SIDE_YEL 0x20
#define SIDE_RED 0x40

/*
 * Initialize pointers to the I/O ports.
 *
 */

unsigned char *LightPort = (char*) 0xbf900000;
unsigned char *Reset     = (char*) 0xbfa00000;

int InterruptHandler()
{
    /* Add code here! */
}

/*
 * The main program initializes the timer port for interrupts
 * and the state, installs the interrupt routine in the standard
 * interrupt routine, enables timer interrupts, and then goes
 * into a waiting loop for as long as switch number 1 is on.
 *
 */

int main()
{
    init_ext_int();
    install_normal_int(InterruptHandler);
    enable_int(EXT_INT3);
    while (*LightPort & 1) {}; /* Loop as long as switch 0 is 1 */
    disable_int(EXT_INT3);
    return 0;
}

Add code to this program stub and construct a working program to control the traffic lights. The traffic lights should follow the standard sequence for a two-way crossing with fixed scheduling, where both roads have equal priority and there are no car sensors in the ground. The red-green light states should last longer than the other states. You can control the LEDs by writing bit patterns to the LightPort as shown above, and the InterruptHandler routine will be called once for every timer interrupt. InterruptHandler should check that it was the interrupt EXT_INT3 that started it, and if so, return a non-zero value. Note that the file interrupts.s must be added to the project, since it contains the definitions of install_normal_int. Note that you must use interrupts for the timing of the traffic light.

Home Assignment 4

Write another version of the traffic light controller, where the lights are green for the main road until a car arrives on the side road (a button is pressed, which triggers an interrupt), at which time the side road should turn green for a while. After that, the traffic light should switch back to green for the main road.

A Simple I/O Program in C

In Laboratory Exercise 1, Home Assignment 2, we saw how a simple I/O task could be solved in assembly language. You will now see how this can be even more simply solved in a high-level language.
Assignment 1
Create a new project, type in, save and build the C program of Home Assignment 1. Execute the program and verify that it works correctly.

Compare the assembly program of Laboratory 1 and the C program. Is there any difference in complexity? Compare the assembly program of Laboratory 1 and the assembly generated by the C compiler. How similar are they?

Traffic Light Control
You will now use a traffic light control task for a larger example of interrupt-driven programming in a high-level language. Included in the lab hardware is a little road crossing equipped with red, yellow and green lights, just as in a real traffic crossing. The task is to write the control program that would be located in the control computer for the traffic lights.

Assignment 2
In the first version of the control program, we assume that the crossing has no sensors in the road and should just switch back and forth between letting the main and the side road having green. The “green” states should last longer than the intermediate switching states, but otherwise, the traffic lights should follow a simple sequence over and over again.

Create a project, type in, save and build the program of Home Assignment 3. Execute the program, debug it if needed and verify that it works correctly.

Assignment 3
In the second version of the control program, we assume a crossing equipped with a car sensor in the pavement of the side road. This light stays with green on the main road until the sensor tells that a car has arrived on the side road. Then the program starts the switching sequence, lets the car across and then goes back to the resting state with green on the main road. The car sensor is simulated by one of the buttons on the lab board.

Create a new project and make a copy of the program of Home Assignment 3. Modify the program to handle a car sensor on the side road, debug it and verify that it works correctly.

Conclusions
Before you pass the laboratory exercise, think about the questions below:

■ What are the relative merits of assembly language and C in larger programming projects?
■ Which is easier to debug, a “standard” program or an interrupt-driven one?
■ Could the traffic light control be solved well by polling? Explain.
■ Are there interrupt-driven programs in real traffic lights?
■ What is the most difficult and time-consuming task in software development?