ECE253 - Digital and Computer Systems

University of Toronto

Labs 9 and 10: Subroutines, Stacks, I/O devices, and Interrupts

The goal of this lab exercise is to continue to gain familiarity with assembly language programming, to learn more about subroutines, and to communicate with an I/O device by using memory-mapped I/O, and interrupts.

Part I

Write an assembly language program to sort a list of 32-bit unsigned numbers into descending order. The first 32-bit number in the list gives the number of items in the list and the remainder of the entries are the numbers to be sorted. The list of data must be sorted "in place", meaning that you are not allowed to create a copy in memory of the list to do the sorting. The list you should sort, including the number count, can be defined using the **.word** directive at the end of your program as follows:

List: **.word** 10, 1400, 45, 23, 5, 3, 8, 17, 4, 20, 33

Note that the first number in the list is 10, indicating that there are 10 numbers following it. In your code you should have a main program that loops through the list of numbers as needed to perform the sorting operation. Use a simple algorithm, such as a bubble sort. Provide pseudo code, similar to C code, that illustrates how your sorting algorithm works. Then, write an assembly language program.

As your main program loops through the list of numbers it will be necessary to compare the values of pairs of list elements to see if they need to be swapped. You are to use a subroutine, called SWAP, to compare such list elements. The SWAP subroutine is passed the address of a list element, compares it to the following element in the list, and swaps the two elements in memory if necessary. The SWAP subroutine should return 1 if a swap is performed, and 0 if not. Pass the address of the first list element to SWAP in register R0, and also pass the return value back to the main program in register R0.

Part II

Write a program that turns on one LEDR light at a time on the DE1-SoC board. First, the light $LEDR_0$ should be on, then $LEDR_1$, then $LEDR_2$, and so on. When you get to $LEDR_9$, the direction should be reversed. Only one LEDR light is ever on at one time. The effect should be a single light sweeping from right-to-left, then left-to-right, and so on.

Use a delay so that the light moves at some reasonable speed. To implement the delay, use the MPCORE Private Timer described in the lectures. When KEY_3 is pressed, the sweeping motion should stop. Pressing KEY_3 again should restart it.

To synchronize with the timer device in your main program use polled I/O by repeatedly reading the *F* bit in the timer's status register. Also use polled I/O to deal with the pushbutton *KEYs*.

Part III

In Part II you used the timer to create a delay, and used polled I/O to synchronize with the timer. Also, you used polled I/O to check when KEY_3 was pressed.

In this part of the exercise, you are to create the same application, but relying entirely on interrupts instead of polled I/O. Interrupts have to be generated by both the timer and KEY port.

An outline of the main program that you need to use for this part is shown below.

```
        .text
        .global  _start
_start:
        . . . initialize the IRQ stack pointer . . .
        . . . initialize the SVC stack pointer . . .

        BL       CONFIG_GIC                  // configure the ARM generic interrupt controller
        BL       CONFIG_PRIV_TIMER           // configure the MPCore private timer
        BL       CONFIG_KEYS                 // configure the pushbutton KEYs

        . . . enable ARM processor interrupts . . .

        LDR      R6, =0xFF200000             // red LED base address
MAIN:
        LDR      R4, LEDR_PATTERN            // LEDR pattern; modified by timer ISR
        STR      R4, [R6]                    // write to red LEDs

        B        MAIN

/* Configure the MPCore private timer to create interrupts every 1/10 second */
CONFIG_PRIV_TIMER:
        LDR      R0, =0xFFFEC600             // Timer base address
        . . . code not shown
        MOV      PC, LR                      // return

/* Configure the KEYS to generate an interrupt */
CONFIG_KEYS:
        LDR      R0, =0xFF200050             // KEYs base address
        . . . code not shown
        MOV      PC, LR                      // return

        .global  LEDR_DIRECTION
LEDR_DIRECTION:
        .word    0                           // 0 means left, 1 means right

        .global  LEDR_PATTERN
LEDR_PATTERN:
        .word    0x1
```

Store your main program in a file, for example *main.s*. Create three other files, called *exceptions.s*, *timer_ISR.s*, and *key_ISR.s*. The *exceptions.s* file should initialize the exception vector table and provide code for all exception handlers. It should also include the subroutine *CONFIG_GIC*, which initializes the ARM Generic Interrupt Controller

The file *timer_ISR.s* is the MPCORE Private Timer interrupt service routine. It has to modify the global variables *LEDR_DIRECTION* and *LEDR_PATTERN* that are declared in the main program above. The first of these variables specifies if the LEDR light is currently sweeping to the left or to the right, and the second variable sets which light is currently turned on. Finally, the file *KEY_isr.s* is the pushbutton KEY interrupt service routine. It has to stop and restart the timer when KEY_3 is pressed.

Skeleton assembly language files are provided on the course BlackBoard with the lab writeup. Fill in the missing parts of these files to create your program.

Preparation and In-Lab Marks

Preparation marks: Write the required pseudo code and assembly language programs. **Note:** your programs have to be commented well enough to enable another person to understand them. The preparation for Parts I and II are due in the first week of the lab, and Part III is due in the second week of the lab.

In-lab marks: You are required to demonstrate to a TA all three parts of the lab exercise. Parts I and II are due in the first week of the lab, and Part III is due in the second week of the lab. For Part I you should use the Monitor Program to illustrate your sorting algorithm—place a breakpoint after each call to the SWAP subroutine, and display the Memory tab so that you can observe the sorting of the list. For Parts II and III you need to be able to demonstrate the sweeping pattern of the LEDs and show that you can start and stop it using KEY_3. For Part III you should demonstrate that you can set a breakpoint in an interrupt service routine.