COMP 231
Lab 6

Rhodes College
—1848—

For this lab exercise, you will use a provided NIOS II CPU system. Your program must work on the provided Nios system and Quartus project. **You only need to submit your assembly-language source file**.

Programming with Functions

This lab requires a functioning assembly-language implementation of the SSD lookup table from the last lab. Please contact me if you were unable to get the previous assignment working correctly.

For this assignment, you will implement a Nios II assembly language that uses an external runtime library that follows the Nios calling conventions. Your program must adhere correctly to the conventions in order to work with the library code. The library contains the `_start` label where execution begins and will call your `main()` function, having initialized devices and created the stack for you.

```
1   int fib(int n) {
2     if (n <= 1) {
3       return n;
4     } else {
5       return fib(n-1) + fib(n-2);
6     }
7   }
8
9   int main(void) {
10     int val, res;
11     val = read();       // read 8-bit value from switches
12     res = fib(val);
13     printHex(res);      // print 32-bit value to SSDs in hex
14     _printResult(res); // print 32-bit value to LCD display in decimal
15     return 0;
16  }
```

To complete this lab, you will need to implement the following instructions using the proper Nios calling conventions and stack frames:

- **int main(void)** – This is the `main()` function in the program listed above. You should use the label `main:` for this function, the `_start` label is defined by the library code. You can assume that the runtime library has setup the stack for you and initialized its own internal state.

  The main function calls three functions that you must implement: `read()`, `fib()`, and `printHex()`. It also calls the `_printResult()` function which is provided by the runtime library. Since `main()` is both a callee (from `_start`) and a caller, you must follow the calling convention rules that may apply.

- **int read(void)** – This function reads an 8-bit value from the switches and returns it to the main function. This function should be very short.

- **int fib(int n)** – The C code for this function is listed above. Your function must adhere to calling conventions. Since it is a recursive function, you *must* save the values of n to the stack in some fashion.

- **void printHex(int val)** – This function takes a 32-bit value and displays it in hexadecimal on the bank of eight SSDDs. This function relies on a helper function, `printNybble()`, which writes the hex character for a 4-bit number to a given SSDD. To implement this function, you should write a loop that takes the next 4-bits from `val` and calls `printNybble()` with it on the next SSDD. The SSDD device addresses start at the value represented by `h0` (see `addrs.s`) and increase by 16-bits for each SSDD.

- **void printNybble(int val, void *address)** – This function takes a 4-bit value and the address of an SSDD. You should adapt your Lab 5 code to take the 4-bit value, lookup the 7-bit SSD value and write it to the device located at `address`.

The `void _printResult(int val)` function takes a 32-bit integer and writes it to the LCD display in decimal. You do not need to implement this. This function is defined in the runtime library, `run.s`. You may examine any assembly code in this file to help you understand correct usage of the calling conventions (and to see how the LCD device works, if you are interested).

Project Setup

For this lab, you are provided a functioning Quartus project with support for all 8 hex displays (SSDDs), the 16x2 LCD display, 8 switches, and 8 LEDs. You are free to look at the Quartus and Platform Designer projects, but if you change any of the device addresses, your lab may not work correctly. The device addresses are stored in `addrs.s` and included into both your lab and the library assembly language files. The leftmost (lowest 4-bits) SSDD is named `h0`, which you can use as a name that can be used with `movia`.

You should write all of your functions in `lab6.s`, leaving `run.s` alone. You will need to create an Altera Monitor Program project for this assignment, just as you did in Labs 4 and 5. You should add both `lab6.s` and `run.s` files to the project. They will be linked together automatically so you can call the `_printResult()` function without issue.

NOTE: You must comment your code. Submitting a correct, but uncommented solution will result in a 25% penalty deduction.

When you have your program working, submit your complete assembly language program to Canvas.