Chenhan Dai, Junchao Zhou
ECE 469 Computer architecture I
May 24, 2023
Lab 5 Report

Procedure

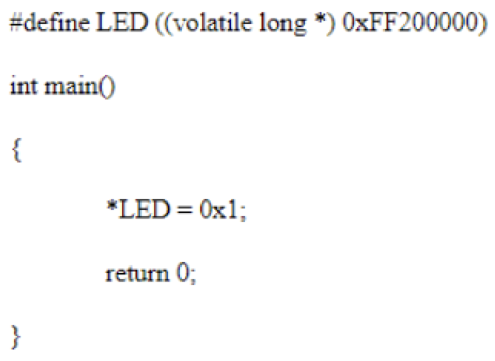
This lab contains 2 tasks: writing and understanding the sample C code for peripheral devices and creating our own C code. And each task contains two parts. The first part for task 1 involves writing a sample C program for LEDs and the second part is related to Switches. The second task consists of creating C code for Pushbuttons and LEDs, as well as comparing the difference between Assembly and C.

Task#1

The first task asks us to search and understand the basic meaning for two sample codes given in the requirements (Figure 1 and Figure 2). We initially conducted an online search to comprehend the fundamental meaning of the two sample codes provided in the requirements. During our exploration, we attempted to modify the variables and observed distinct outcomes. Following this, we engaged in discussions and arrived at our conclusions regarding the questions posed.

Part1:

For this code, it defines a new macro LED and assigns the memory address 0xFF200000 to it. And in the main section, it assigns the value ‘0x1’ to the LED, which illuminates the right most LED. We further conducted experiments by modifying this assigned value, aiming to understand the relationship between different values and the corresponding LEDs' illumination. Based on our observations and analysis, we reached conclusions regarding how varying values affect the illumination pattern of the LEDs and answered the following questions.



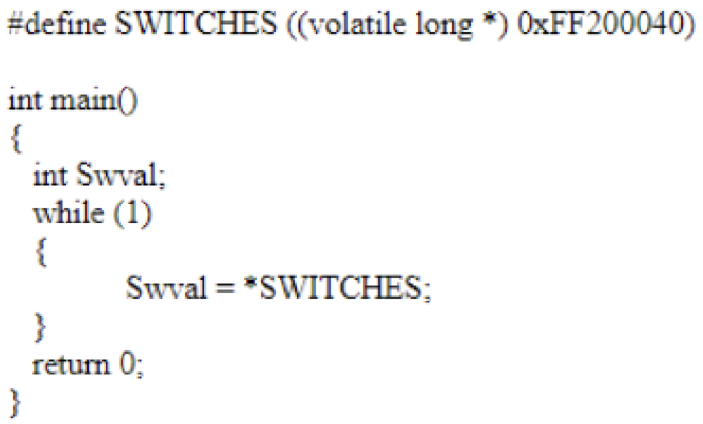
```
#define LED ((volatile long *) 0xFF200000)

int main()

{

        *LED = 0x1;

        return 0;

}
```

Figure 1: sample C program for LEDs

1. What does the (volatile long*) keyword mean?
2. Verify that this program works as intended in the CPUlator. Make sure that the Language set on the CPUlator is in C. Take a screenshot of the LEDs, showing that the right-most LED is illuminated.
3. Modify this example code to illuminate all 10 of the LEDs on the NIOS board. What value did you choose to write to *LED? Why? Take a screenshot of the LEDs, showing that all of the LEDs are illuminated. Make sure to include this screenshot in your report.

Part2:

This code defines a macro "SWITCHES" and assigns the memory address 0xFF200040 to it. In the main function, it uses a while loop to continuously read the value of the memory location "SWITCHES" and stores it in the variable "Swval". We compile the program, and continuously step over the generated Assembly code and understand the meaning of each step. After that, we discussed and answered the following questions.



```
#define SWITCHES ((volatile long *) 0xFF200040)

int main()
{
  int Swval;
  while (1)
  {
          Swval = *SWITCHES;
  }
  return 0;
}
```

Figure 2: sample C program for Switches

1. Compile this program. In the Disassembly tab, continuously Step Over the generated Assembly code until you are looping through the main branch.
2. What is the PC address of the main branch? Why is it not at address 0?
3. Why is this while(1){} loop necessary?
4. Which temporary registers (R0, R1, R2, etc.) hold the value of Swval?
   R2.
5. Notice that when we do a Step Over in the main branch, the temporary register R2 toggles between three different hexadecimal values. What's the significance of these three different values that you see?

Task#2

Part 1:

For this part we were asked to write our own C code to read values from pushbuttons and reflect it into the corresponding LEDs. Following questions are given to guide us to create our own code to interact with NIOS.

1. What is the register address for the LEDs on the NIOS processor? Taking a look at the example codes in Task 1, how would we define this register address in C?

2. What is the register address for the Pushbuttons on the NIOS processor? Taking a look at the example codes in Task 1, how would we define this register address in C?

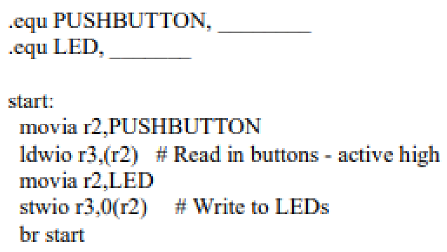
3. How do we obtain (in C) the information on which button was pressed?

4. When you compile your program, what is the size of the ELF executable (you can find this information in the Messages section of the CPUlator)?

5. Take a screenshot of the LED and Pushbutton peripherals on CPUlator and include them in your report. Also save your .c source code (Ctrl + S, or go to File -> Save) as Lab5_task2_part1.c.

Part 2:

In this part, the assembly code of previous part is given portionally by guidelines. We are asked to fill the blank part of code to make it runs.



```
.equ PUSHBUTTON, ________
.equ LED, _______

start:
  movia r2,PUSHBUTTON
  ldwio r3,(r2)   # Read in buttons - active high
  movia r2,LED
  stwio r3,0(r2)    # Write to LEDs
  br start
```

And following questions is given to guide us to use and understand the code

1. Fill in the register address for the PUSHBUTTON and LED. Compile this program on the CPUlator. For this, make sure to set the Language to Nios II. Verify that the functionality for this assembly code is the same as the C code you created in Task 2 part 1.

2. When you compile this program, what is the size of the ELF executable (you can find this information in the Messages section of the CPUlator)?

3. Is the size of this ELF executable similar or different from the size produced from the C code? Please explain thoroughly why these are similar or different.

4. Save your assembly code (Ctrl + S, or go to File -> Save) as a .s file called Lab5_task2_part2.s.

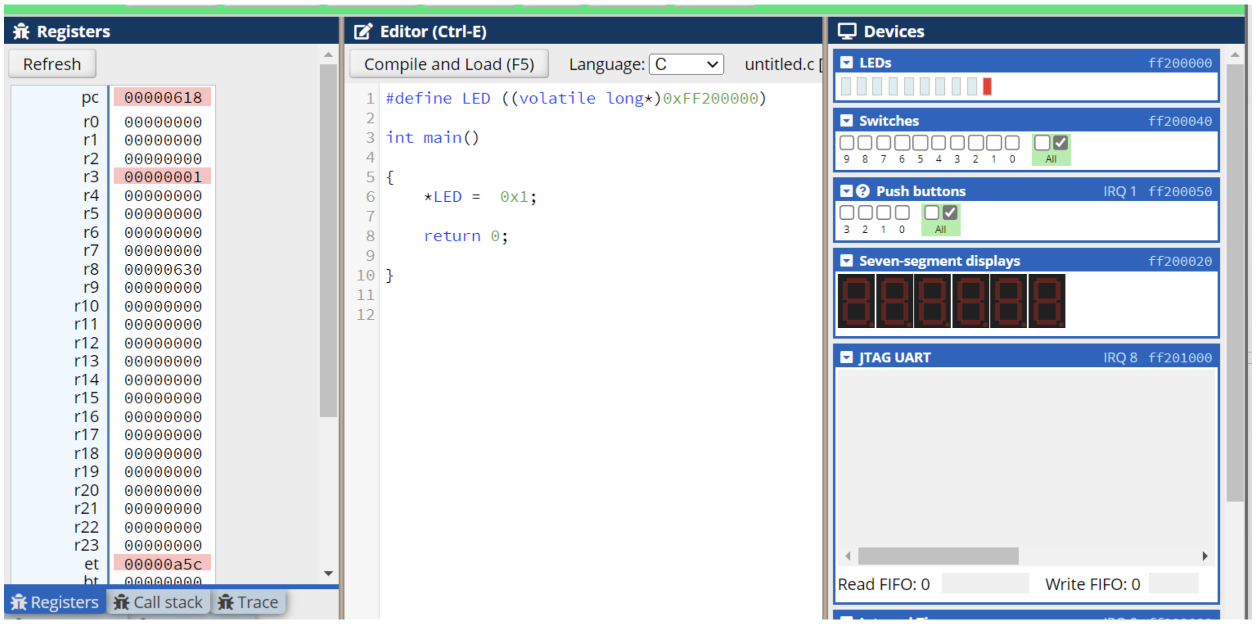
Result:
Task #1
Part 1: Sample C Program for LEDs

We wrote the following sample code and answered a few questions.

```
#define LED ((volatile long*)0xFF200000)

int main()

{
        *LED =  0x1;

        return 0;

}
```

1. What does the (volatile long*) keyword mean?
The (volatile lone*) keyword is a typecast in C programming language. It is used to explicitly convert a pointer to a 'long' type into a pointer to a 'volatile long' type.

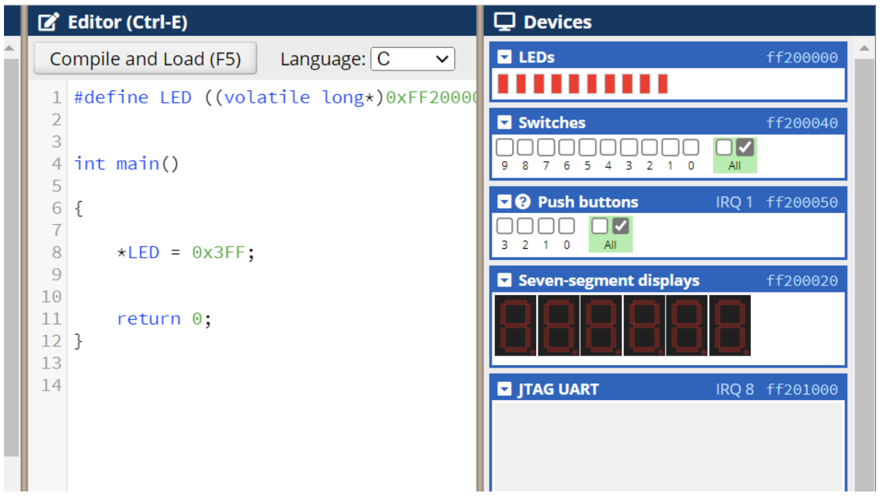
2. Verify that this program works as intended in the CPUlator. Make sure that the Language set on the CPUlator is in C. Take a screenshot of the LEDs, showing that the right-most LED is illuminated.



```
pc   00000618
r0   00000000
r1   00000000
r2   00000000
r3   00000001
r4   00000000
r5   00000000
r6   00000000
r7   00000000
r8   00000630
r9   00000000
r10  00000000
r11  00000000
r12  00000000
r13  00000000
r14  00000000
r15  00000000
r16  00000000
r17  00000000
r18  00000000
r19  00000000
r20  00000000
r21  00000000
r22  00000000
r23  00000000
et   00000a5c
bt   00000000
```

```c
#define LED ((volatile long*)0xFF200000)

int main()

{
    *LED =  0x1;

    return 0;

}
```

3. Modify this example code to illuminate all 10 of the LEDs on the NIOS board. What value did you choose to write to *LED? Why? Take a screenshot of the LEDs, showing that all of the LEDs are illuminated. Make sure to include this screenshot in your report.
I choose to write 0x3FF as it represents the binary number ‘1111111111’, where each bit corresponds to an LED. It sets all the bits to ‘1’ and turns on all the LEDs.



```c
#define LED ((volatile long*)0xFF20000


int main()

{

    *LED = 0x3FF;


    return 0;
}
```

Part 2: Sample C Program for Switches

And then we wrote the following code and answered a series of questions.
#define SWITCHES ((volatile long*)0xFF200040)

```
int main()
{
   int Swval;

   while (1)
   {
      Swval = *SWITCHES;
   }

   return 0;
}
```

1. Compile this program. In the Disassembly tab, continuously Step Over the generated Assembly code until you are looping through the main branch.

2. What is the PC address of the main branch? Why is it not at address 0?
The PC address of the main branch is 230. It is not at address 0 as the initialization code resides at the lower memory address.

3. Why is this while(1){} loop necessary?
The while(1){} loop is an infinite loop, which repeatedly reads the value of the switches and assigns it to the variable "Swval".

4. Which temporary registers (R0, R1, R2, etc.) hold the value of Swval?
R2.

5. Notice that when we do a Step Over in the main branch, the temporary register R2 toggles between three different hexadecimal values. What's the significance of these three different values that you see?
There are three different hexadecimal values: FF200000, FF200040 and 00000000. FF200000 is the initial state of "Swval" , and FF20040 is the memory address for Switches, and the last value is the value read from "SWITCHES". We could manually change the switches to change this value.

Task 2: Creating your own C code
Part 1: C Code for Pushbuttons and LEDs

In this part, we wrote our own C code to read the values of the NIOS pushbuttons and displayed that value onto the LED peripheral.

Answer the questions below to help guide you through this process.

1. What is the register address for the LEDs on the NIOS processor?
   0xff200000

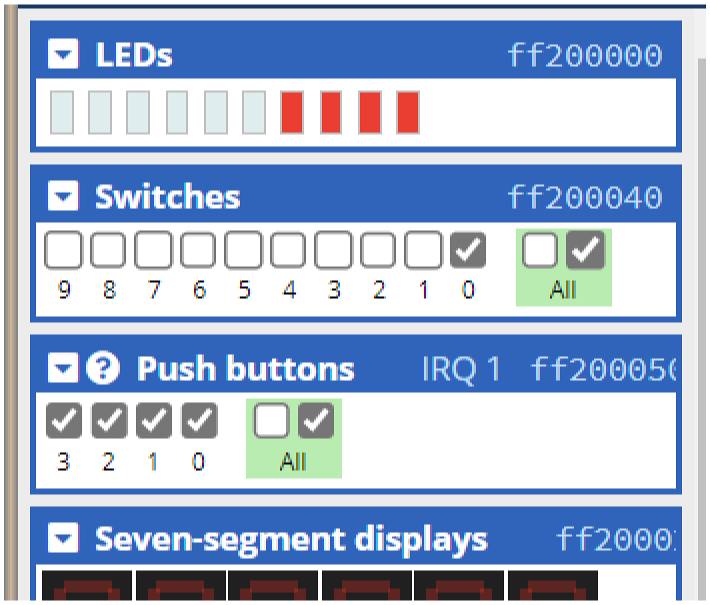
2. What is the register address for the Pushbuttons on the NIOS processor?
   0xff20050

3. How do we obtain(in C) the information on which button was pressed?
   We created a new variable and stored the value from the Pushbuttons and converted it to the LEDs.

4. When you compile your program, what is the size of the ElF executable?
   The size is 2565 bytes.

5. Take a screenshot of the LED and Pushbutton peripherals on CPUlator and include them in your report.

```
#define Pushbuttons ((volatile long*)0xFF200050)
#define LED ((volatile long*)0xFF200000)

int main()
{
   int Puval;

   while (1)
   {
          Puval = *Pushbuttons;
          *LED = Puval;
   }

   return 0;
}
```



Part 2: Assembly vs C

Below is the code for Task 2, part 1 written in NIOS II Assembly (minus the register address for LEDs and Pushbuttons).

```
.equ PUSHBUTTON, 0xff200050
.equ LED, 0xff200000

start:
        movia r2, PUSHBUTTON
        ldwio r3, (r2) #Read in buttons - active high
        movia r2, LED
        stwio r3, 0(r2) #Write to LEDs
        br start
```

The size of the ELF executable is 28 bytes, which is different from the C code. And the reason is C code is a higher-level language which generates additional machine code to handle things like function calls, memory management and variable declarations. Besides, C code requires initialization and startup code to set up the program environment.

Appendix:

Task1 Part1

```
#define LED ((volatile long*)0xFF200000)

int main()

{
        *LED =  0x1;

        return 0;

}
```

Task1 Part2

```
#define SWITCHES ((volatile long*)0xFF200040)

int main()
{
   int Swval;

   while (1)
   {
      Swval = *SWITCHES;
   }

   return 0;
}
```

Lab5_task2_part1.c

```
        #define Pushbuttons ((volatile long*)0xFF200050)
        #define LED ((volatile long*)0xFF200000)

        int main()
        {
           int Puval;

           while (1)
           {
               Puval = *Pushbuttons;
               *LED = Puval;
           }

           return 0;
        }
```

Lab5_task2_part2.s

```
.equ PUSHBUTTON, 0xff200050
.equ LED, 0xff200000

start:
        movia r2, PUSHBUTTON
        ldwio r3, (r2) #Read in buttons - active high
        movia r2, LED
        stwio r3, 0(r2) #Write to LEDs
        br start
```