Lab #4 - Input and Output Organization

ELEE 3450U: Microprocessors & Computer Architecture
Fall 2022

Huzaifa Zia (100779087)
Hanzalah Patel (100785622)
Zubair Islam (100778152)
Waddah Saleh (100785692)
Nathan Hacault (100778576)

Thursday, November 17th, 2022

Introduction

The purpose of this lab is to better understand the input/output ability of a processor. The two methods being used in the lab are program-controlled polling I/O technique and interrupt-driven approach. A parallel port interface will be used on the Altera DE series board. The first part of the lab asks to use LEDs within the program to display the accumulated sum. The second part of the lab task asks for a push button to be added to a program for users to press when they want to read a new number into a program that reads numbers continuously. The third part asks to modify a given code to display a sum as a hexadecimal number in the display.

Part 1

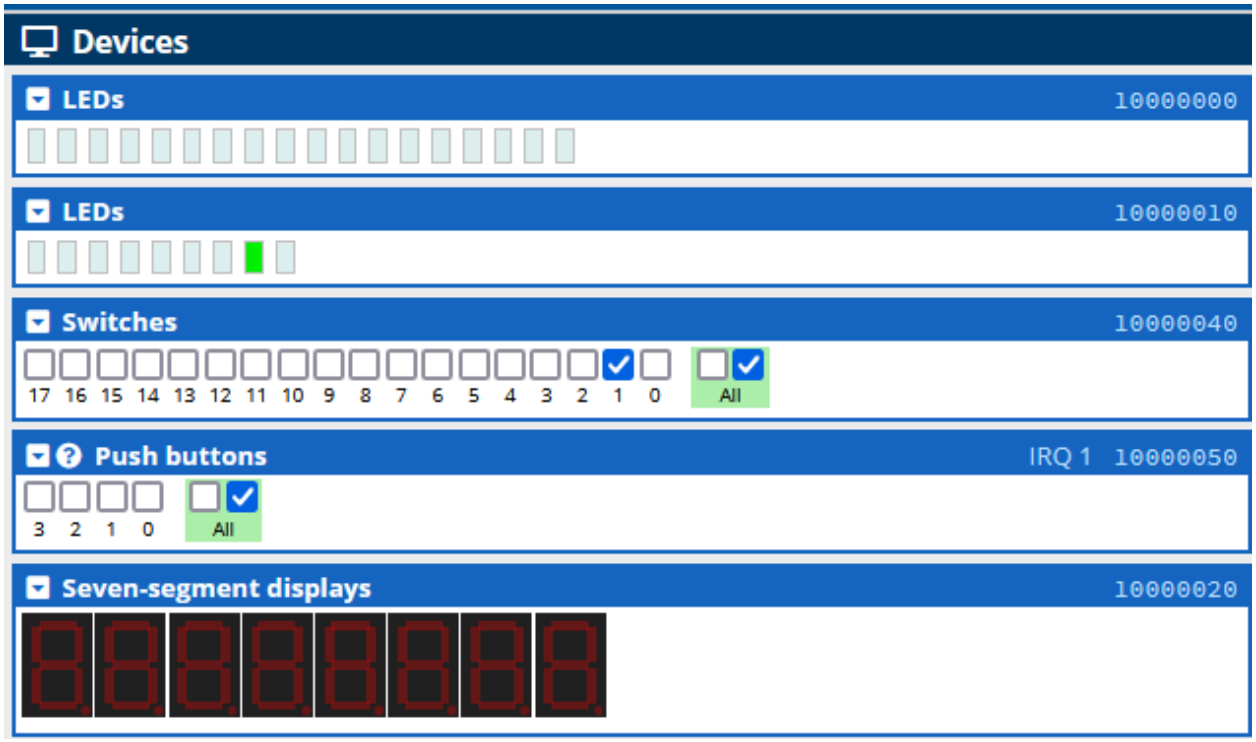

Figure 1: After clicking on switch 1 and stepping into multiple times, LED lights up in position 2.

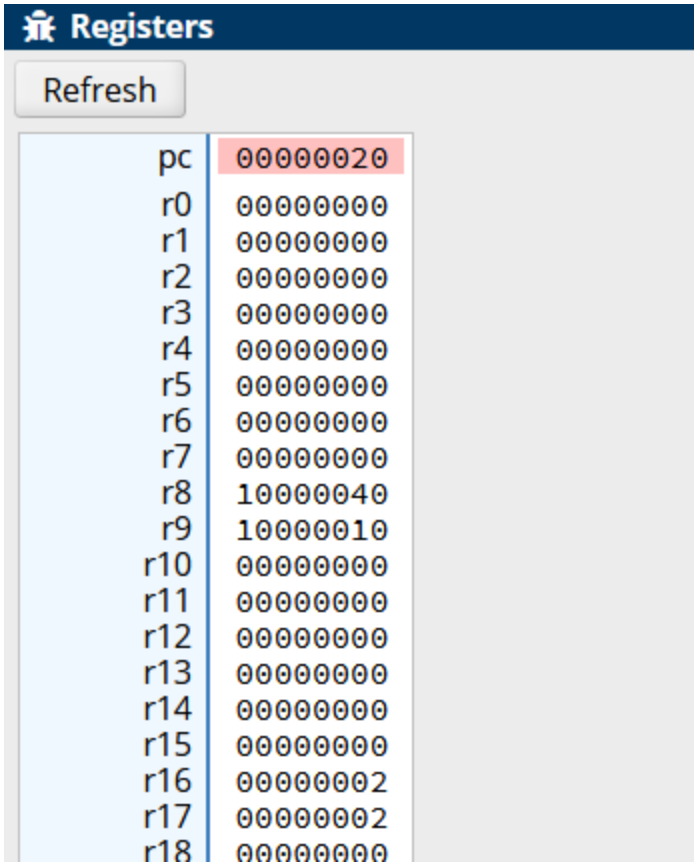

Figure 2: Registers after stepping into switch 1 on. r16 holds switch num and LED holds sum

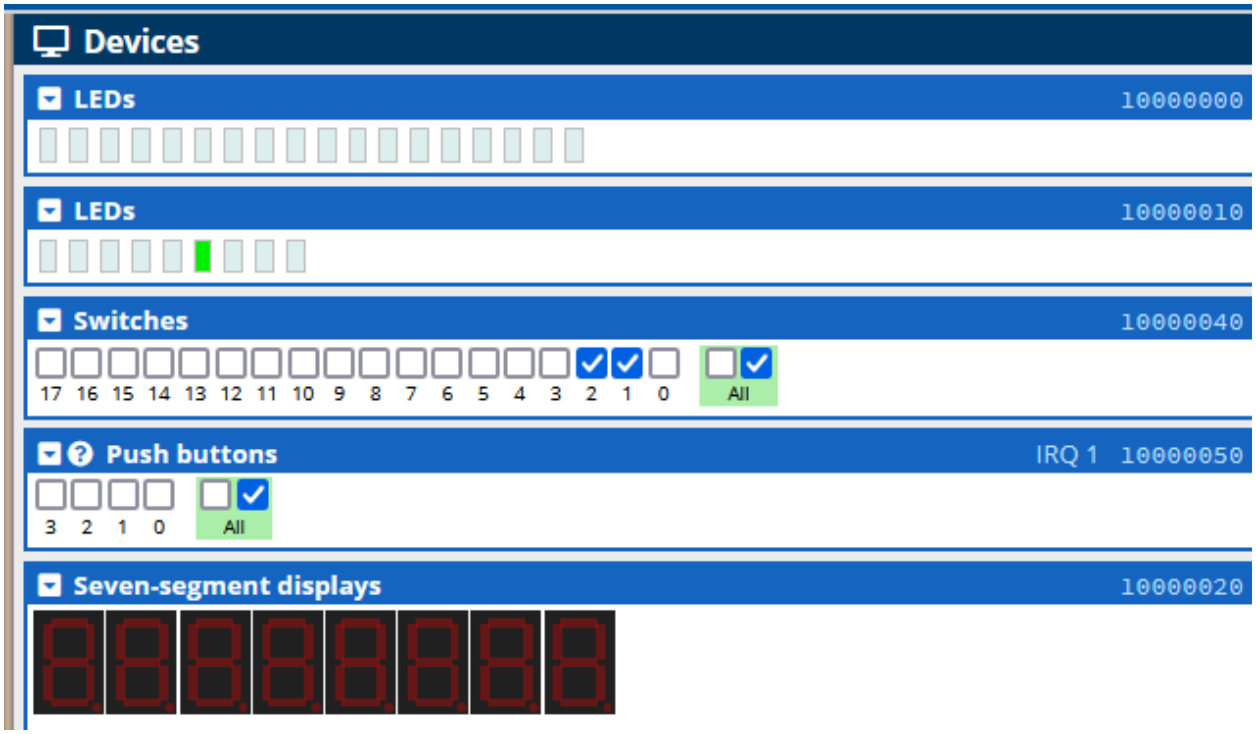

Figure 3: After clicking on switch 2 and keeping switch 1 on and stepping into multiple times, LED lights up in position 4.

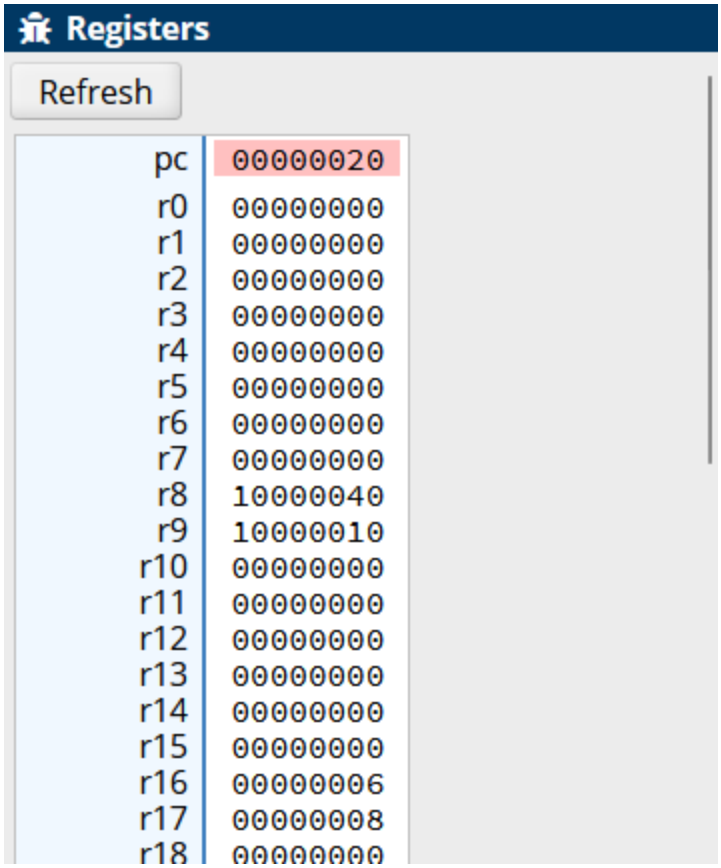

Figure 4: Registers after stepping in with switches 1 and 2 on. r16 holds switch num and LED holds sum on gd and bd.

Part 2

```
/* r17 - The accumulated sum                                                  */
/* r18 - The status read from Pushbuttons port                                */
/****************************************************************************/
.global _start
_start:
    add r17, r0, r0         /* Clear the sum register                */

    movia   r8,  NEW_NUMBER         /* Set up the switches address           */
    movia   r9,  GREEN_LEDS         /* Set up the green LED address          */
    movia   r11, Pushbuttons        /* Set up the Pushbuttons address        */


LOOP:
    ldwio   r16, 0(r8)          /* Read in the new number                */

    ldwio   r18, 12(r11)            /* Read the Edge-capture register        */
    andi    r18, r18, 0x2           /* Check if KEY1 was pressed and     */
    beq r18, r0, LOOP           /*   branch back if not              */
    stwio   r0,  12(r11)            /* Clear the Edge-capture register       */

    add r17, r17, r16           /* Add the new number to the sum         */
    stwio   r17, 0(r9)          /* Display the new sum                   */

    br  LOOP                /* Loop to continue polling          */

.end
```

Editor (Ctrl-E) | Disassembly (Ctrl-D) | Memory (Ctrl-M)

Figure 5: Filled in blanks

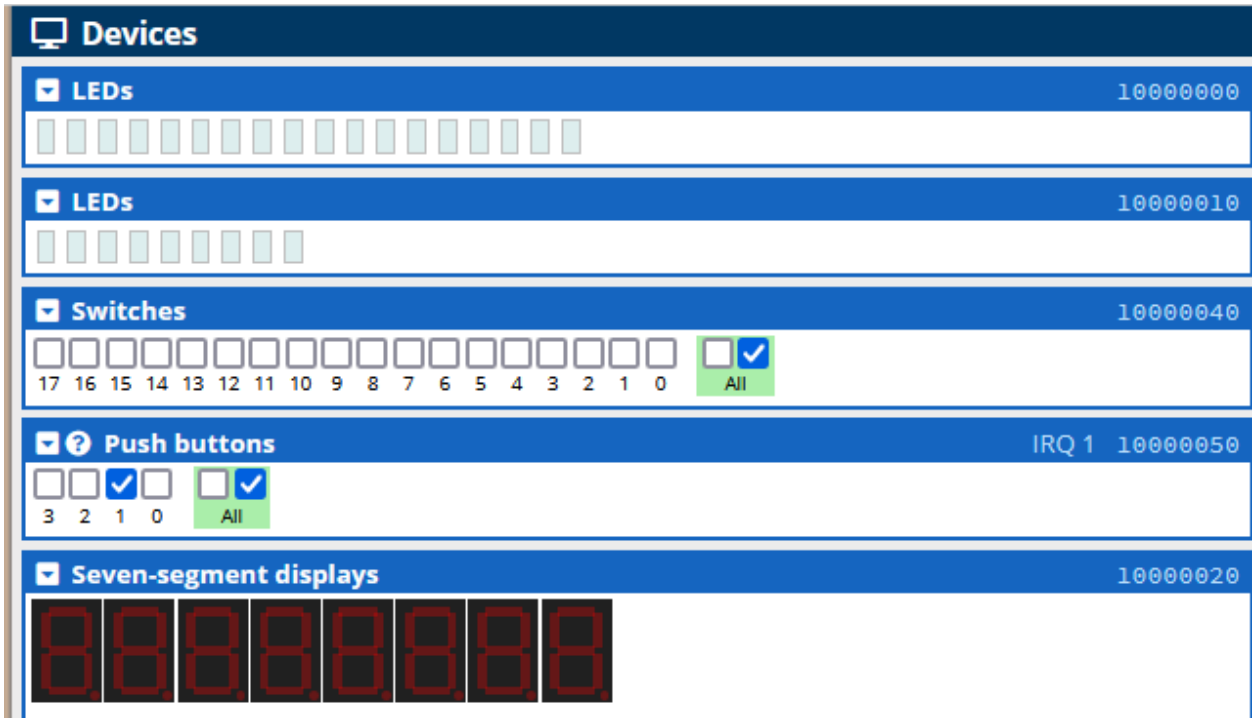

Figure 6: Selecting push button 1

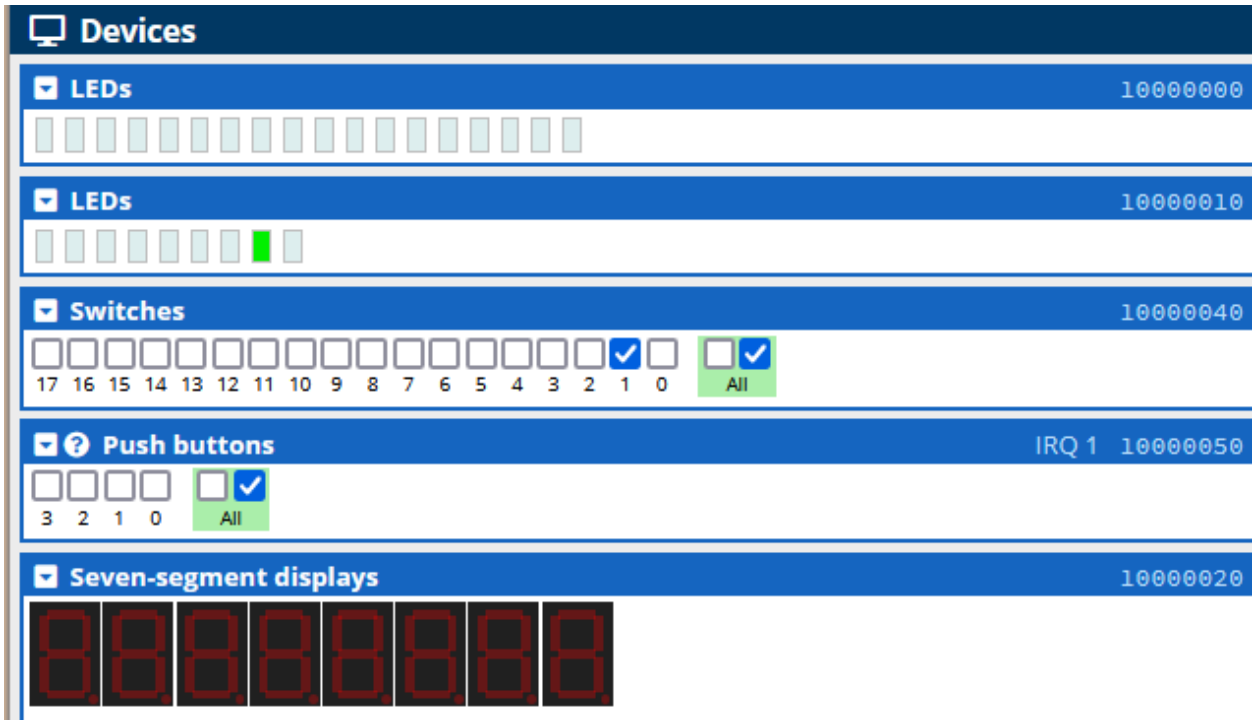

Figure 7: After deselecting push button 1 and selecting switch 1 and stepping, LED is lit up in the 4th position which is the accumulated sum of 8.

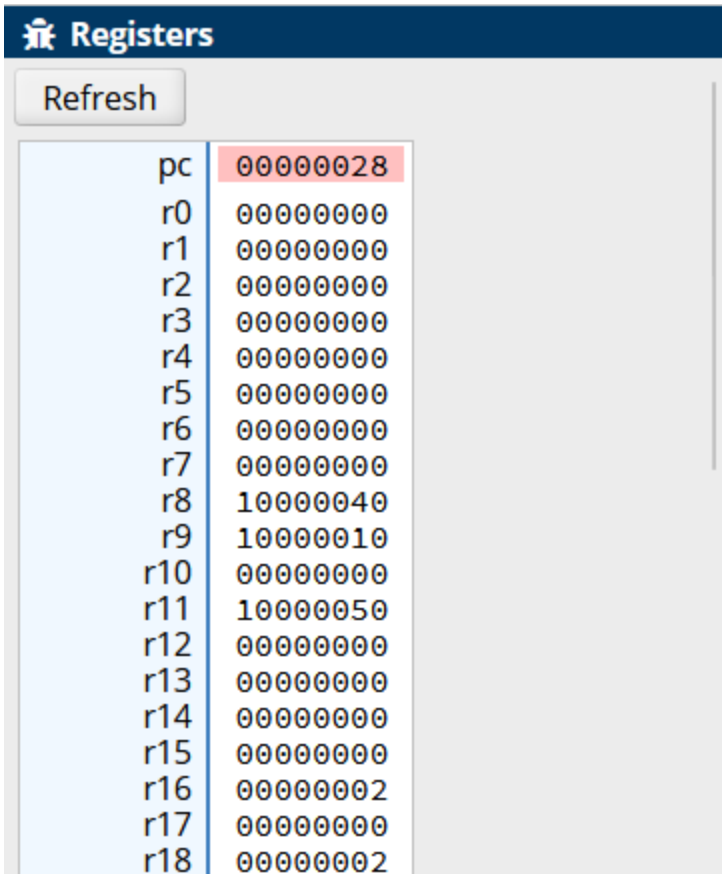

Figure 8: After selecting switch 1 from above picture on gd, R16 holds 2 which is switch value on gd and r18 holds 2 which is push button.

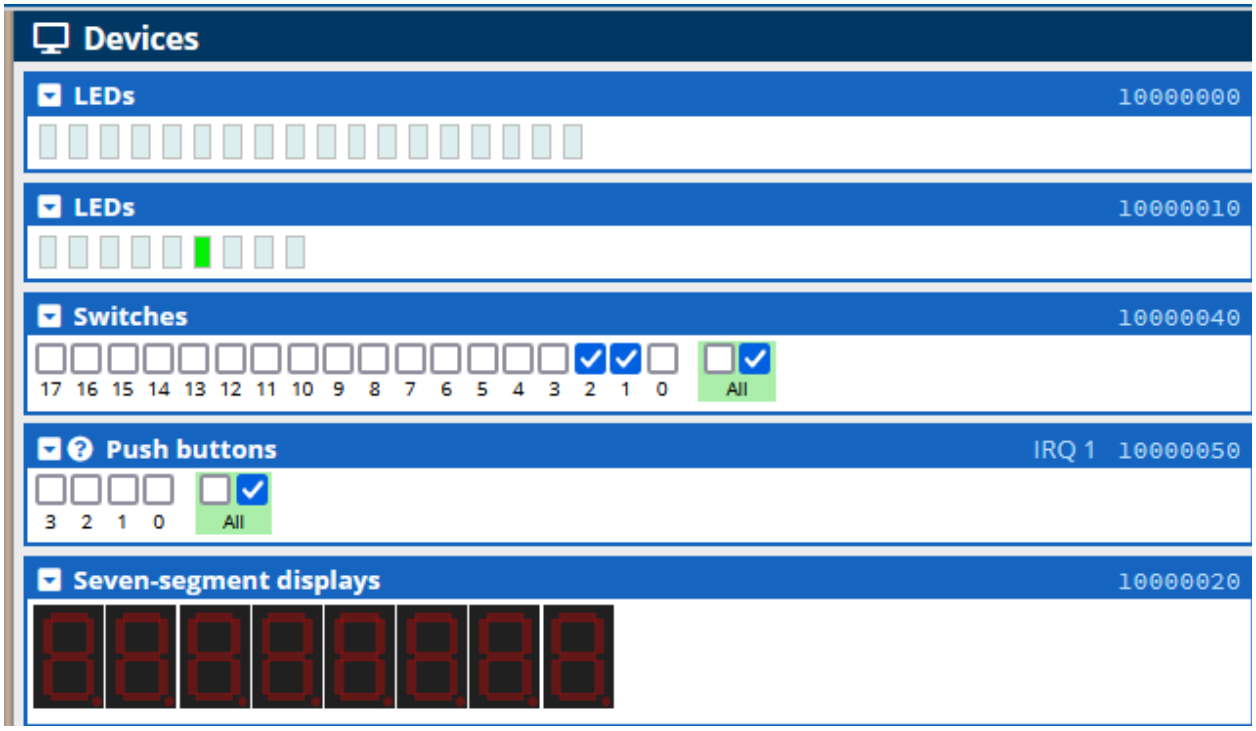

Figure 9: After selecting switch 2 and single stepping, LED is lit up in the 4th position which is the accumulated sum of 8.

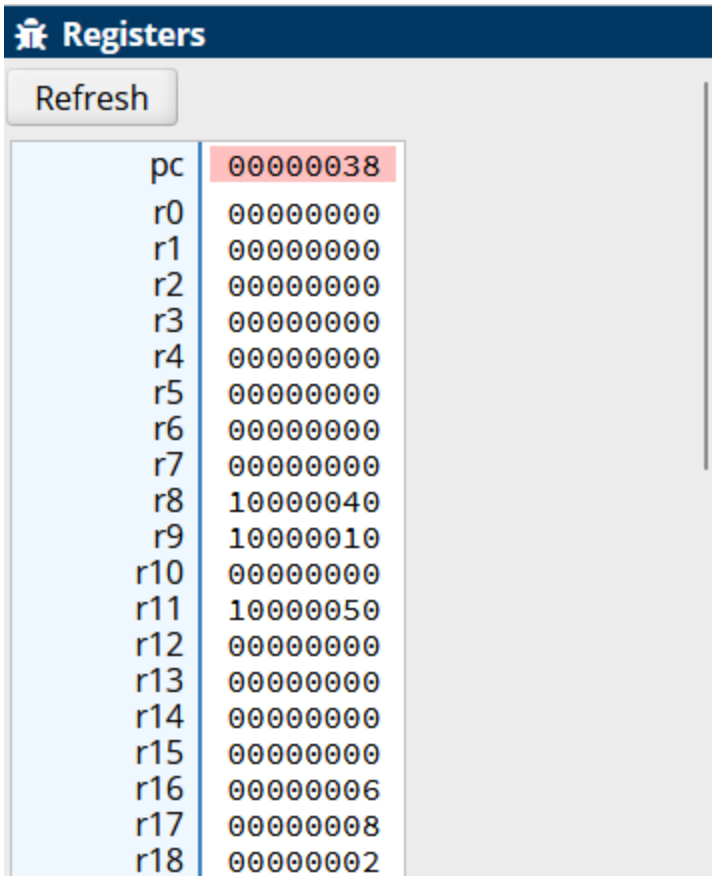

Figure 10: Register info after above picture

Part 3

```
MAIN_LOOP:
    ldwio   r16, 0(r8)          /* Read in the new number                */

    ldwio   r18, 12(r10)            /* Read in the status flag               */
    andi    r18, r18, 0x2           /* Check if KEY1 was pressed and     */
    beq r18, r0, MAIN_LOOP      /*   XXXX   branch back if not                  */
    stwio   r0,  12(r10)            /* Clear the Edge-capture register       */

    add r17, r17, r16           /* Add the new number to the sum         */
    stwio   r17, 0(r9)          /* Display the new sum on green LEDs       */


    /* This part computes the 7-segment display patterns for a given number          */

    add r19, r0, r0         /* Clear r19                             */
    addi    r20, r0, 4          /* Initialize the LOOP2 counter          */
    mov r21, r17            /* r21 holds the number being processed  */
LOOP2:
    andi    r22, r21, 0xf           /* Extract a hex digit                   */
    ldb r23, 0x1000(r22)        /* Look up the 7-segment pattern         */
    or  r19, r19, r23           /* Include the pattern in total display  */
    roli    r19, r19, 24            /*    Make room for pattern of next digit   */
    srli    r21, r21, 4         /*    Now consider the next hex digit       */
    subi    r20, r20, 1         /*    Decrement the counter                 */
    bgt r20, r0, LOOP2          /* Branch back if not done               */
    stwio   r19, 0(r11)         /* Display the sum on HEX display        */

    br  MAIN_LOOP           /* Loop to the start of the main program */
```

Figure 11: Replaced XXXX's with corresponding code segments

```
Compiling...
Code and data loaded from ELF executable into memory. Total size is 4112 bytes.
Assemble: nios2-elf-as --gdwarf2 -o work/asmZbc7hy.s.o work/asmZbc7hy.s
Link: nios2-elf-ld --section-start .reset=0  --script build.ld -e _start -u _start -o work/asmZbc7hy.s.elf work/asmZbc7hy.s.o
Compile succeeded.
```

Figure 12: Compilation successful

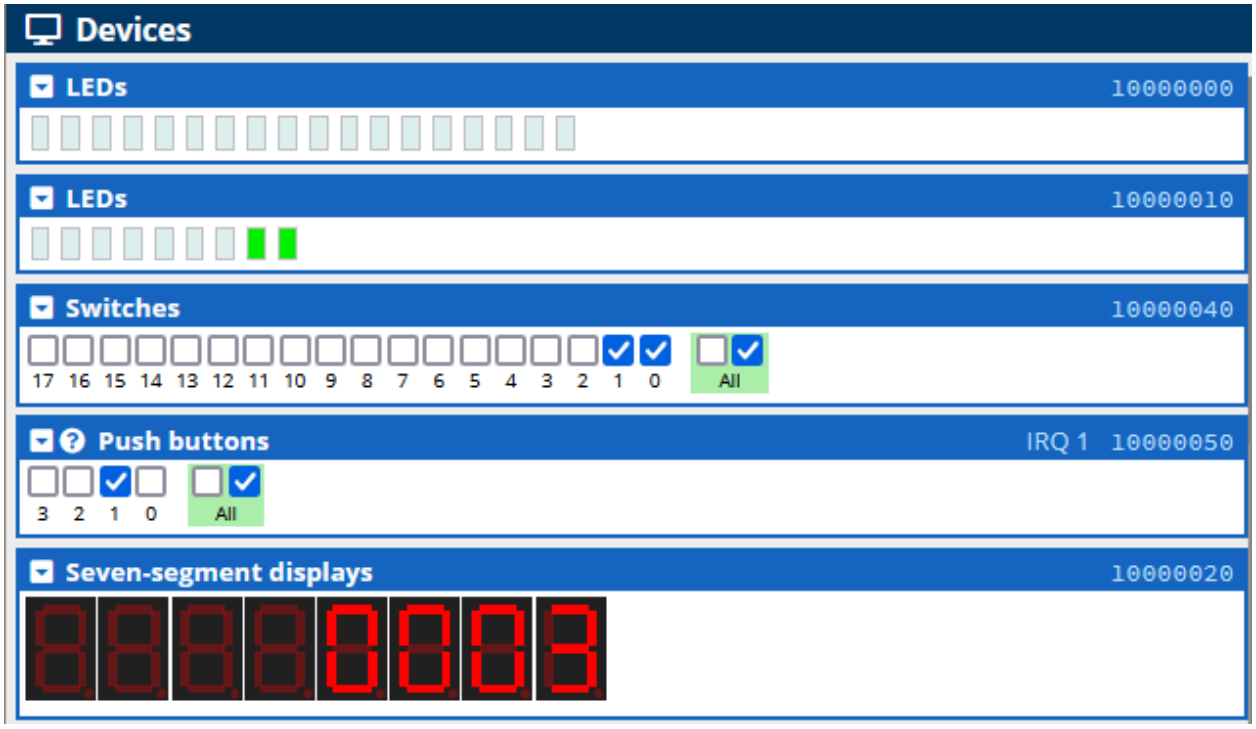

Figure 13: After single stepping through code with switches 11 and push button selected

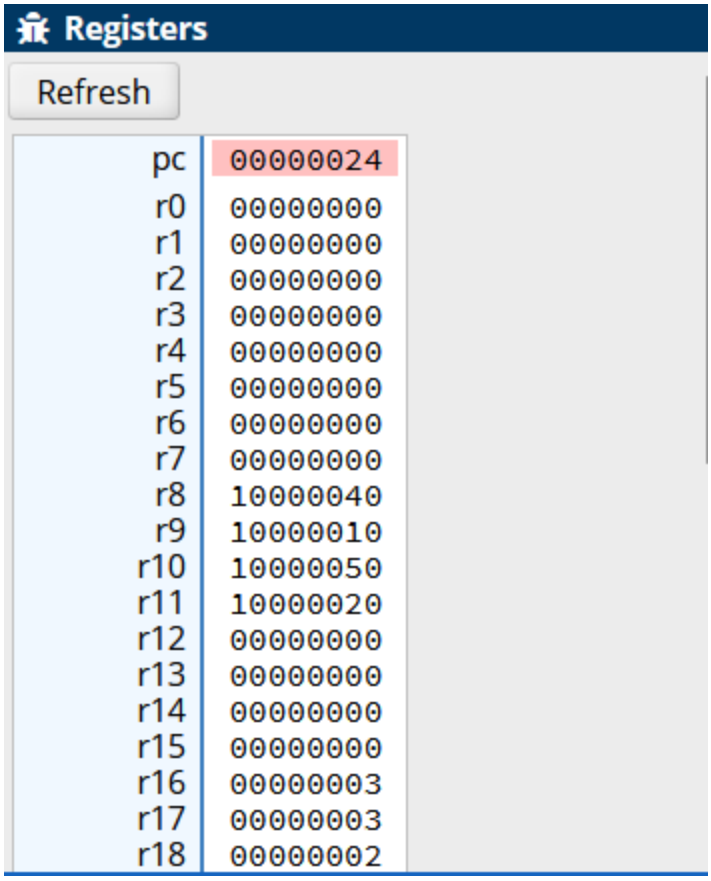

Figure 14: Registers after single stepping through code

Part 4

```
ISR:
    rdctl   et, ctl4
    beq et, r0, ERROR           /* Error if not an external interrupt,   */
                        /*    it is not handled in this example. */
    subi    ea, ea, 4           /* It is a hardware interrupt; decrement */
                        /*   ea to execute the interrupted       */
                        /*   instruction upon return to the      */
                        /*   main program.                       */
    ldwio   r15, 12(r10)            /* Read Edge-capture register            */
    andi    r15, r15, 0x2           /* Check if KEY1 was pressed         */
    beq r15, r0, ERROR          /* Error if bit 4 is not equal to 1  */
    stwio   r0, 12(r10)         /* Clear the edge-capture register to    */
                        /*   prevent the same interrupt request  */
                        /*   from being serviced again.          */

MAIN_LOOP:
    ldwio   r16, 0(r8)          /* Read in the new number                */

    add r17, r17, r16           /* Add the new number to the sum         */
    stwio   r17, 0(r9)          /* Display the new sum on green LEDs     */

/* This part computes the 7-segment display patterns for a given number             */
    add r19, r0, r0         /* Clear r19                             */
    addi    r20, r0, 4          /* Initialize the LOOP2 counter          */
    mov r21, r17            /* r21 holds the number being processed  */
LOOP2:
```

Figure 15: Replaced XXXX's with corresponding code segments

```
Compiling...
Code and data loaded from ELF executable into memory. Total size is 1296 bytes.
Assemble: nios2-elf-as --gdwarf2 -o work/asmKmEomO.s.o work/asmKmEomO.s
Link: nios2-elf-ld --section-start .reset=0  --script build.ld -e _start -u _start -o work/asmKmEomO.s.elf work/asmKmEomO.s.o
Compile succeeded.
```

Figure 16: Compilation successful

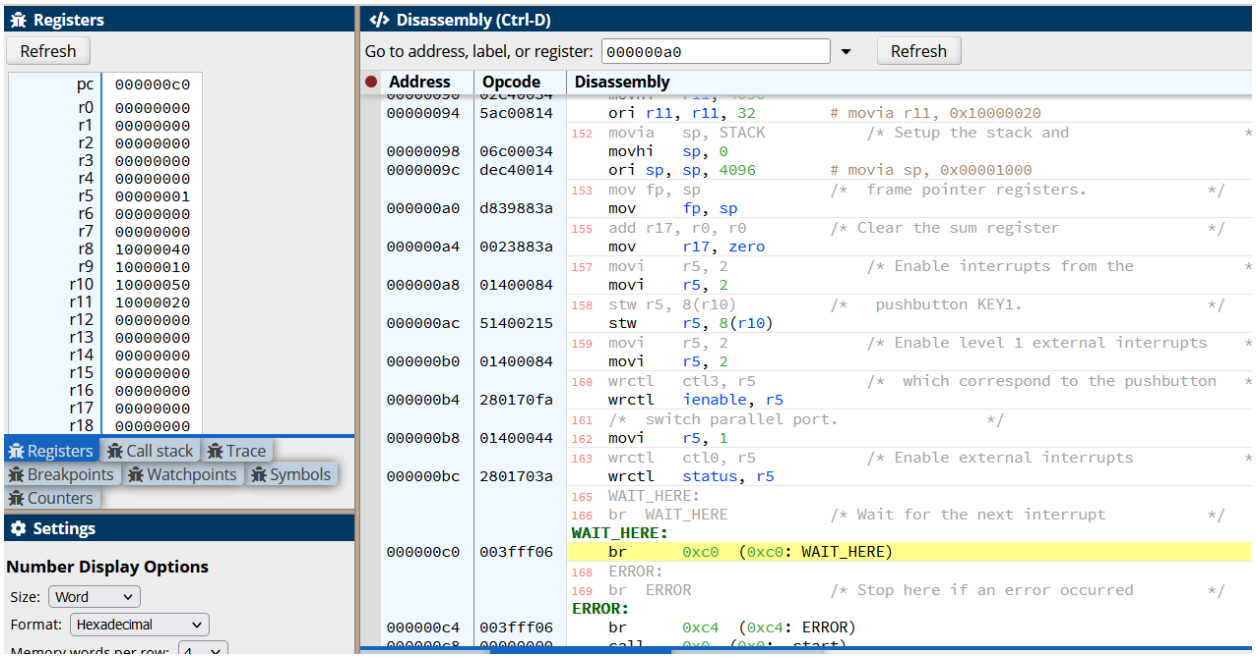

Figure 17: The program waits here until the interrupt signal is received

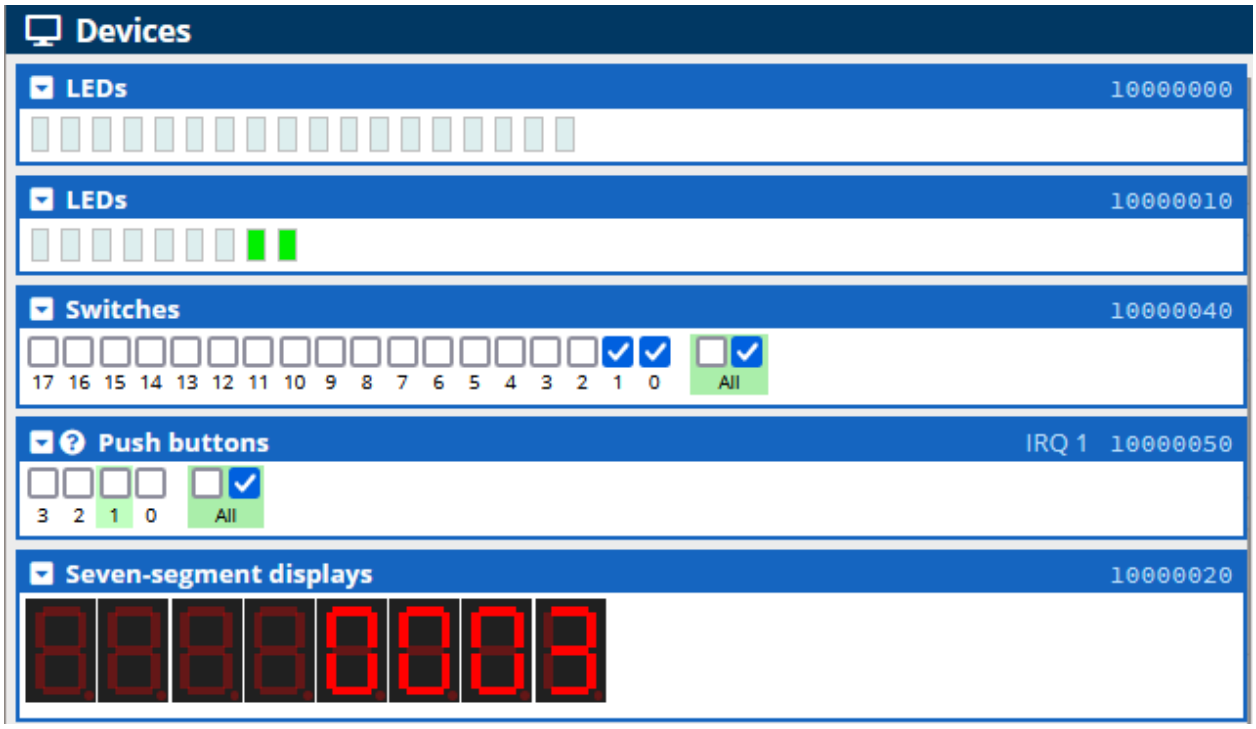

Figure 18: Stepped into the program after activating and deactivating the push button and selecting two switches.

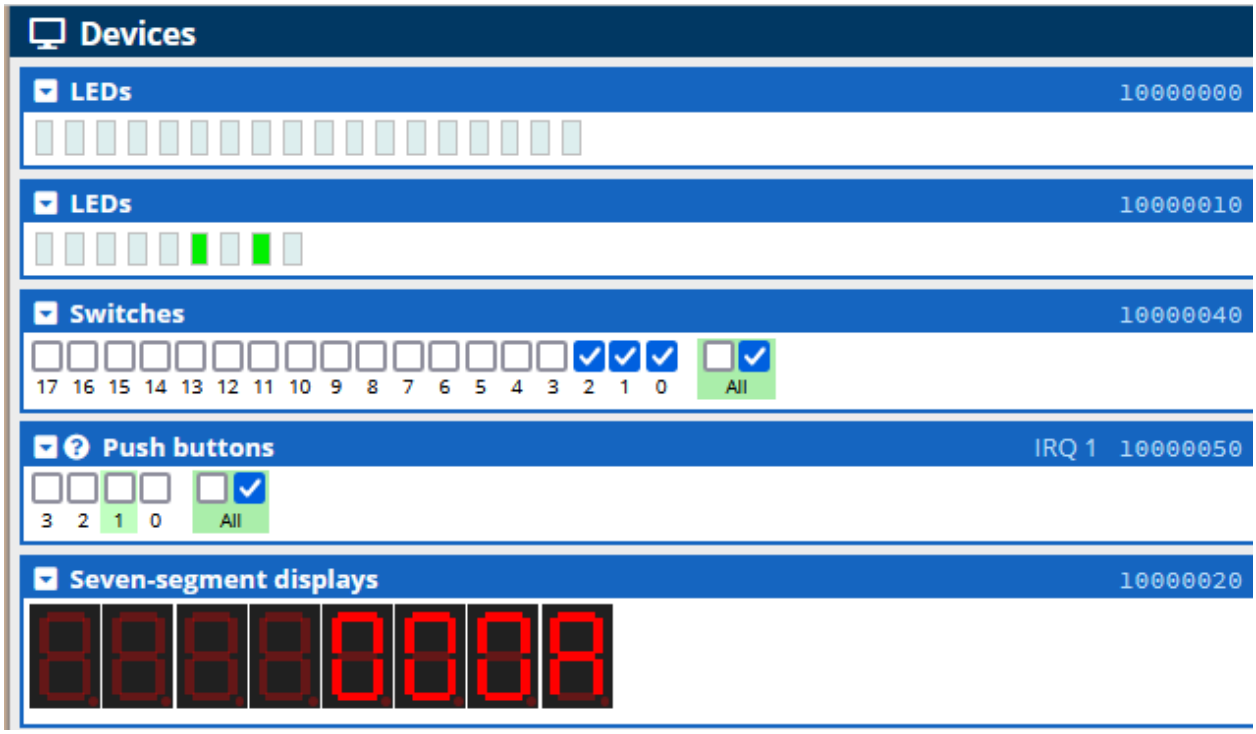

Figure 19: Selecting a different value and calculated a sum

Deliverables:

1. In part 1, is the "new number" read from memory? If not, where does it read from?

No, the number is read from the switches. The new number is sent to the processor using the input switches. These are connected through parallel ports and connected directly to register 16.

2. In part 2, how to modify the source code so the Key2 can be utilized?

Instead of using the line *andi r18, r18, 0x2* we would use *andi r18, r18, 0x4*

3. How many registers does the PIO interface contain? What are they?

The PIO interface contains 4 registers. They are the data, direction, interrupt-mask and edge capture registers.

4. In part 3, which register is used in the IO polling? Which line of code is used?

R8 is used to store the address of the data to retrieve. R10 holds the address of pushbutton, 3 registers further used for Edge-Capture/status register, These values are stored in r16 and r 18 respectively to check for further behavior.

Code:

```
ldwio r16, 0(r8)
ldwio r18, 12(r10)
```

5. In part 4, describe the interrupt method in your own words.

The interrupt method disrupts the normal execution of a program and causes the execution of special instructions. When the interrupt occurs, the microcontroller saves whatever it is currently doing and executes corresponding ISR code.

Conclusion

The lab was a success as all lab sections were completed successfully, and the group's understanding of polling input and output devices with a program controlled method and an interrupt driven input and output management method was enhanced. Furthermore, the group also learned more about Programming Input Output (PIO) Interfaces that is composed of four different registers; Data, Direction, Interrupt-mask, and Edge-capture register. The registers were used in different ways throughout each part of the lab to display the sums on the LED and seven segment display. The parallel port interfaces were used to accomplish these tasks.

Thus, this lab demonstrated fundamental and important topics of computer architecture and contributed to the improvement of the group's understanding of input and output mechanisms in the computer.