

**HIGH-LEVEL LANGUAGES FOR MICROCONTROLLERS**

**Lab#02**

**Student name:**

**Student Number:**

**Lecturer name:**

**Lab instructor:**

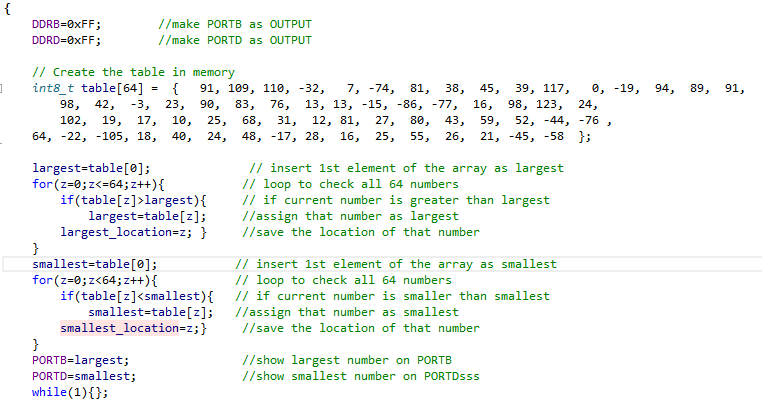
**Subject name:**

**Introduction:**

In this lab we have developed the code for finding largest and smallest number in an array in high-level language C. In last lab we have made the same code using assembly language and observed its behavior using assembler and debugger. The basic aim of this lab to differentiate between the codes of the C and assembly. Although the functionality of both codes(C and assembly) is same but we have to look the size, number of instructions, memory view and other things as well.

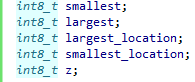
**Discussion:**

**Design implementation of the code:**

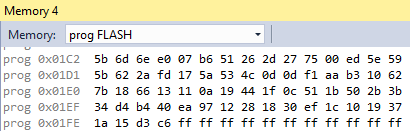
****

The above pic shows the whole code for finding largest and smallest numbers in array. Firstly we have declared PORTB and PORTD as output ports to show largest and smallest numbers. After that the table of 64 numbers is declared. Further two variables named as smallest and largest as declared as global to save the largest and smallest number. Also two variables named as largest location and smallest location has been declared to save the location of largest and smallest number in table. Then we have started a loop which starts from 0 and in this loop we are comparing the values of table one by one with the largest variable. If the table value is greater than table value than the variable largest value then the new value has been assigned in that variable and its location is saved in the largest\_location variable. Else the old value remains the largest value. In this ways 64 values of the table are compared and largest number and its location is found. Same logic is performed for finding smallest value and its location. In the end the, the both largest and smallest value are shown on the PORTB and PORTD respectively. To keep the code, running again and again we made a loop while(1) which keeps the control to it forever so that code cannot execute again and again.

**Explanation of variables used, and table location in Memory?**

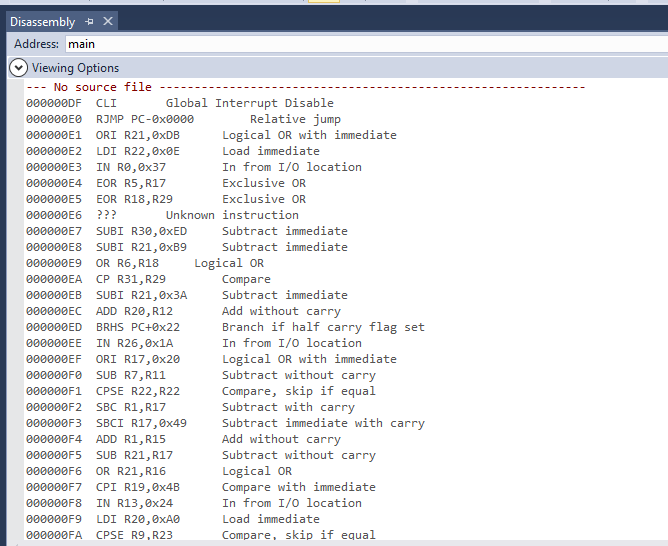
****

We have declared all the variables as uint8\_t which means that all the variables are taken as the 8 bit int type and so each variable will take the 8 bits of memory and also in some variables the signed values will be stored and in some variables the unsigned values will store. The largest variable will save the largest value while the smallest will save the smallest. Also there location will also be saved.

s

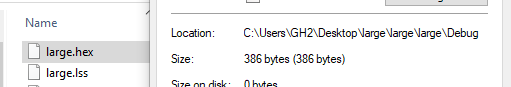
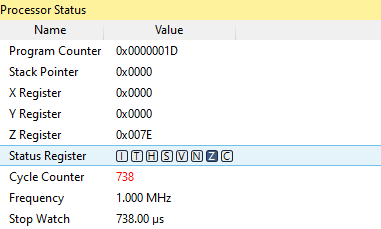
The location of our table is shown in the memory and its states that it is starting from 0x01C2 location and ending at location 0x0202.And overall it is consuming 64 bytes in memory. All the numbers are represented in Hexadecimal.

**Disassembled C code, execution time, Clock Cycles required, and code size?**

****

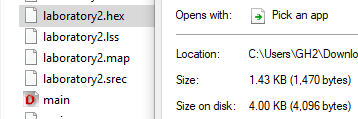
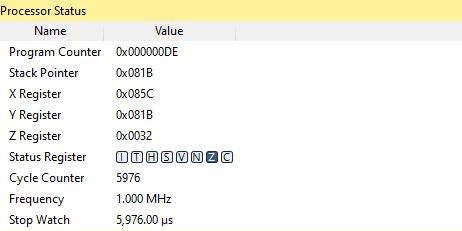
The above pic shows the disassembled code. There are many instructions in this code which has been automatically added the complier in this assembly code and this assembly code contains so many instruction as compared to our own made assembly code. Due to this reason the size of this c code is greater than the lab1 code.

**Clock cycle and execution time of lab1 code:**

****

**Execution Time, Cycle counter File size after compilation**

**Clock cycle and execution time of C code:**

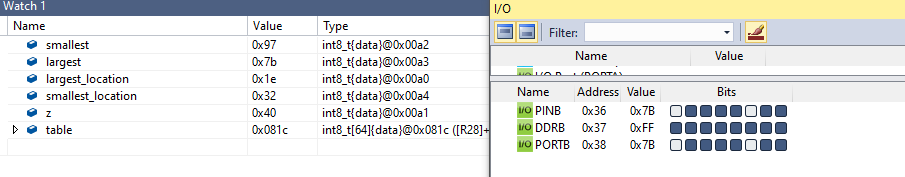
****

**Execution Time, Cycle counter File size after compilation**

The above comparison shows the total numbers of cycles taken by both codes and their execution time on 1MHZ frequency. The assembly code takes 738 clock cycles so its execution time is 738us while the C code takes the 5976 cycles so it takes 5079us. Also the size of assembly code is 386 bytes and C file is 1.43Kb. So, overall the assembly code is memory efficient and as well as fast to execute.

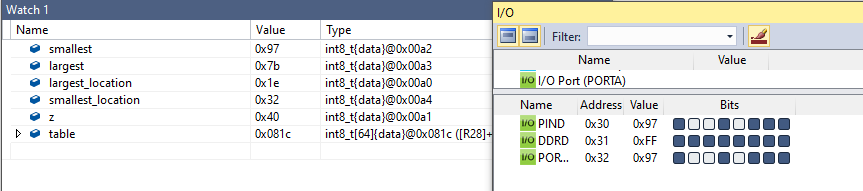
**Output Results:**

**For Largest number:**

****

The above fig shows the output for the largest value. The variable largest on the left hand side is having value 7b (123) in it and its location in table is shown as 1e (30) in the largest location variable. On the right hand side we have shown that value on PORTB. The PORTB is containing 01111011.

**For smallest number:**

****

The above fig shows the output for the smallest value. The variable largest on the left hand side is having value 97 (-105) in it and its location in table is shown as 0x32(50) in the smallest location variable. On the right hand side we have shown that value on PORTB. The PORTB is containing 10010111b.

In this part the smallest number is to be find. It means the more the number will be negative the more it will become small. And we also knows that computer stores all the negative numbers in 2’s complement form. So in our final smallest we will see the 2’s complemented result.

For example if smallest number is -105 so

105=0110 1001 (0x69)

Taking 1’s complement becomes

1001 0110(0x96)

Adding 1 for taking 2’s complement

1001 0110

+ 1

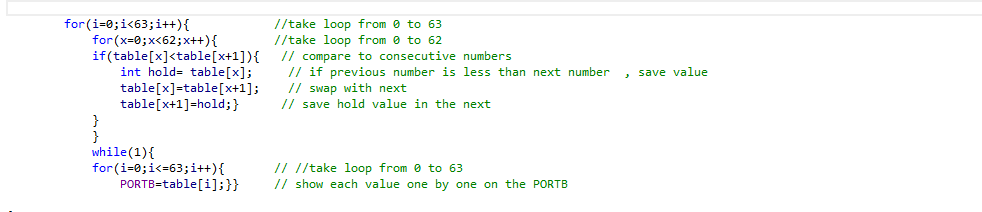
1001 0111 (0x97)

Hence in output we will see -105 as 0x97

**(i)**

**Table elements is descending order and showing on PORTB:**

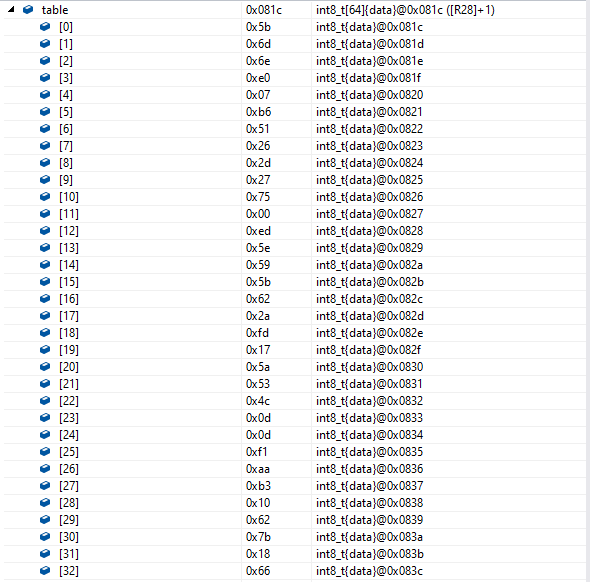
**Code Explanation:**

****

The code shown above orders the array by the method of bubble sort. In this method we have to compare each element of the table with the next value of the table if the value is less than the next value than we swap both numbers and move on in this way the whole 64 values are sorted. For this purpose the hold variable is used temporarily to hold the value of one variable during swapping.

And in the last we loop each value to show on PORTB and this process of showing continues infinitely as it is placed in while (1).

**Results showing sorted Array:**

****

**Conclusion:**

Hence in this lab we have observed the functionality and behavior of the C code and also we have compared the results, file size, execution time, clock cycle and number of instructions of C and assembly code. And we have assumed from this lab that although C code is more readable for us and easy to write but its hex file is more bulk and processor takes more time to execute the C file as compared to assembly file. On the other hand, assembly file is difficult to read but it is low level language whose hex file is of less bytes and fast to process.

Reference:

1. Microchip, Atmel 8-bit AVR MCU FLASH Microcontroller, "Atmel-8155-8-bit -Microcontroller-AVR-ATmega32A\_Datasheet"
2. Microchip, Atmel 8-bit AVR MCU FLASH Microcontroller, "Atmel-8155-8-bit -Microcontroller-AVR-ATmega32A\_Datasheet", (2020 Reference)
3. Mazidi, Muhammad Ali - "AVR Microcontroller and Embedded Systems: Using Assembly and C", Pearson Custom Electronics Technology, First Edition, 2011,
4. Microchip, Atmel 8-bit AVR MCU FLASH Microcontroller, "Atmel-8155-8-bit -Microcontroller-AVR-ATmega32A\_Datasheet", (2020 Reference)
5. Mazidi, Muhammad Ali - "AVR Microcontroller and Embedded Systems: Using Assembly and C", Pearson Custom Electronics Technology, First Edition, 2011,