

**INTRODUCTION TO ASSEMBLER USING MICROCHIP STUDIO 7.0**

School of Engineering

**EEE2256 - Introduction to Embedded Systems**

Laboratory Task 1 -

Lecturer:

Laboratory Instructor:

Student Name:

Laboratory Session:

Student Number: **s3656848**

Submission Due Date:

Number of .asm files submitted:

**Introduction:**

This lab explains us the thorough understanding of Microchip studio 7.0. By this lab we will learn how to use microchip studio’s built-in debugger and assembler. For this purpose we have used the ATmega32 microcontroller which is an 8 bit microcontroller. This purpose of this lab is to show that how we can implement high level language concepts like loops, if else and other searching algorithms in assembly language. This lab contains two parts. The first part is to find the largest number from a given array and this array contains total 64 elements with both signed and unsigned numbers. The second part is to find the smallest value from that array. Also we have to find the location of that smallest or largest number.

**Background:**

To search for specific element in an array there are many methods which we can be adopted. But each method has its own its limitations and merits. One method is binary search which mostly applies to the sorted arrays. And it is fast than linear search. In this method the whole array is divided into two parts and compared the number with central element. If number is less than central element than it means that our number would be in lower half. So we start searching lower half. Again that lower half is divided into two parts and so on. In that way an array of 1024 elements is searched in only 10 iterations. But the biggest limitation of this method is that it requires sorted array in ascending order. The second method is linear search. In this lab we are using this method as our array is unsorted. In linear search the all the elements of the array are searched one by one. So to search our array in this lab 64 iterations are required.

To find the biggest number in the array the following pseudo code is used:

for i = 0 to 63 do{

if table[i] > largest\_so\_far the{  
 largest\_value\_so\_far = table[i];  
 largest\_value \_index = i;  
 }  
}

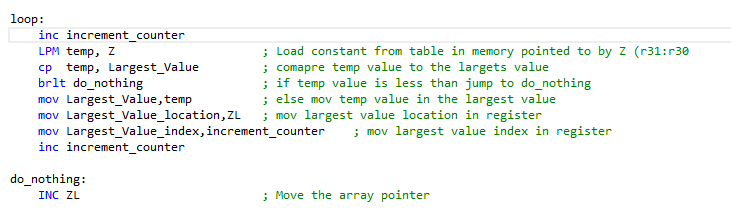
**Linear Search Algorithm to find largest element in Array**

This algorithm is in C language and it is using for loop and if statement to find the largest element in array. In for loop the value of i is kept 0 in start and this loop will run until the value of I becomes 63. So from 0 to 63 this loop will run for 64 times. After entering in the loop the table[i] value is compared with the largest value so far. If table[i] value is greater than it will enter in if statement and the largest value is updated. Also the index of that value is saved. If table[i] value is less than the largest value than only increment in i occurs and next iteration starts.

**Discussion:**

**Design implementation for finding largest number in array:**

The main part of the code is pasted below.



In the starting of the loop, the counter is incremented in start to show the number of iterations and the index of number at which the largest number is present. Then the number is loaded from memory pointed by Z register and that value is stored in temp variable. After that this value in temp is compared with the value already present in the Largest\_value. If temp value is greater than largest\_value then update largest\_value variable with temp value. Also the location and index of that biggest value is saved in Largest\_value\_­location and in Largest\_value\_index via ZL and counter variable. In the end we increment the ZL and next iteration occurs.

**What instruction we are using to load a register from memory address pointed by Z register?**

We are using the **LDI** instruction to the value pointed by Z register into the general purpose resister. In our program this value is going to be stored in R16 register. Z register is 16 bit resister and we can be used as AL and ZH. ZL refers to the lower part of Z and ZH refers to higher part of Z.so by combining both parts the whole address is formed.

**How the Z register is incremented and where it is pointing?**

The **INC** instruction is responsible for incrementing of Z register. After doing each iteration the INC instructions increments 1 in the ZL register. So by doing 1 increment the ZL starts pointing the next number in the table.

**How you will detect the end of the Table?**

CPI ZL, LOW((Tble << 1) + 64)

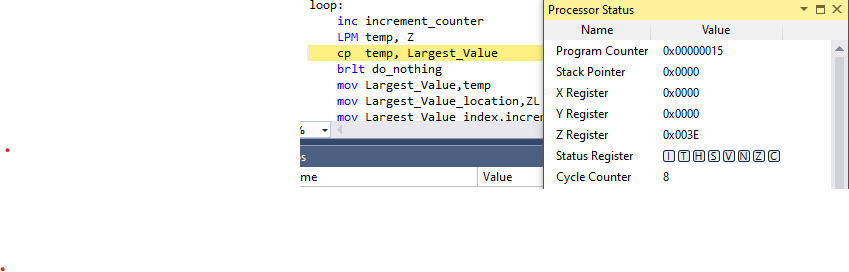
CPI instructions compares the ZL register with the lower byte of the array and comapres either It has reached to te 64 or not. And in the next we use **BRNE** instruction to take the decision either we have to keep in loop or we have to break out. BRNE takes decision on the basis of the subtraction result of above both and watches the Z register. If Z register is zero it means that we have have to keep ourself in loop again. If Z becomes 1 then BRNE instruction will not work and program move forward for infinite loop here.

We have used the **BRLT** instruction which means that Branch if lower than. This instruction comes after the CP instruction. CP instruction compares both of its operands. If left hand operand is greater than right hand operand then this instruction works. BRLT instruction looks for S flag. If S=1 then it branches forward.The are many registers which we have to save our results and indexes. As R16 is used to stored the value pointed by Z. Also R17 contains our result which is greatest value.R18 contains the largest value index and R20 contains the largest value location. R21 contains the increment counter which keeps the record that how many iterations have been occurred.

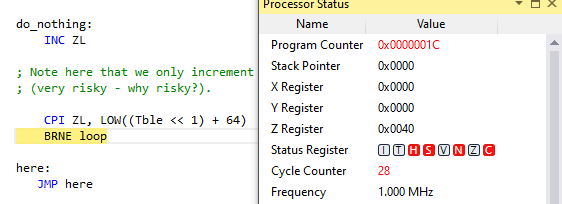
**BRLO** instruction is Branch if less than does the work of comparison but the difference between BRLO and BRLT is that BRLO only works for unsigned numbers while BRLT works for signed numbers. Our array contains both signed and unsigned numbers so we have used BRLT. And we have CP instruction instead of SUB instruction because the reason is that sub instruction performs the subtraction saves result in the destination register and on the basis of this operation flags changes and we can take decision by watching the state of flag. BUT in all the procedure the actual value changes in destination register. This case is different with the CP instructions as it works same as sub instruction but the value of the destination register remain same ass previous only flags changes. So we can take decision without affecting the value of the registers.

**Example When temp value> greatest\_Value:**

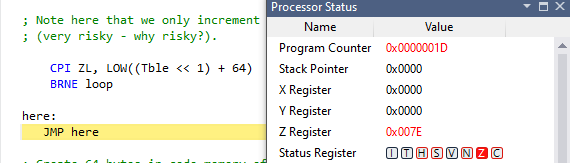
**Status/Flag Regsister:**

****

**Initial Condition**

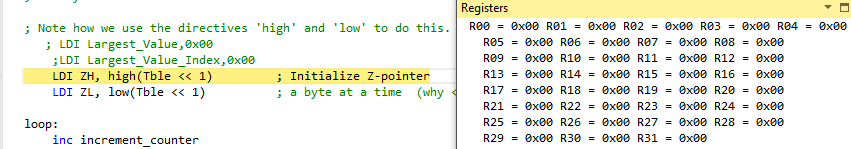
****

**During**

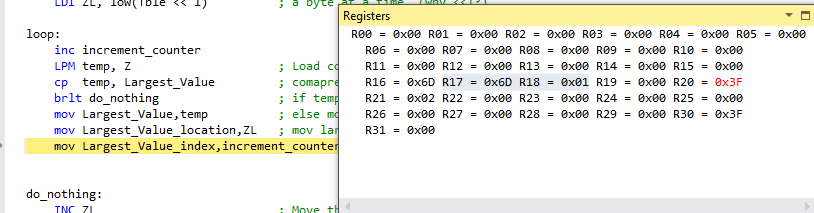
****

**After**

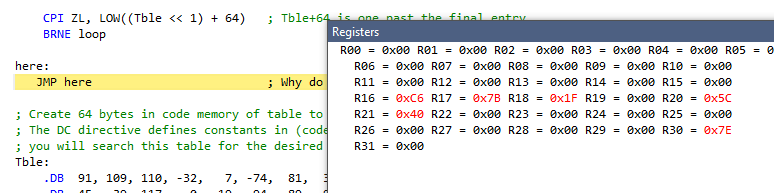
**General Purpose Registers:**

****

**Initial**

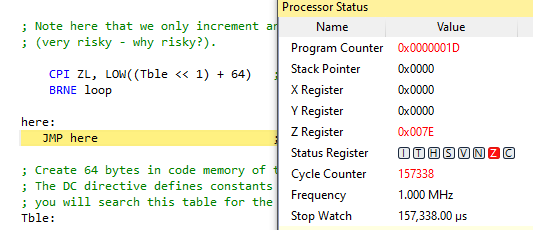
****

**During**

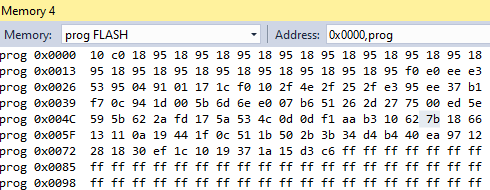
****

**After**

**Final Result when program jumps to here:**

****

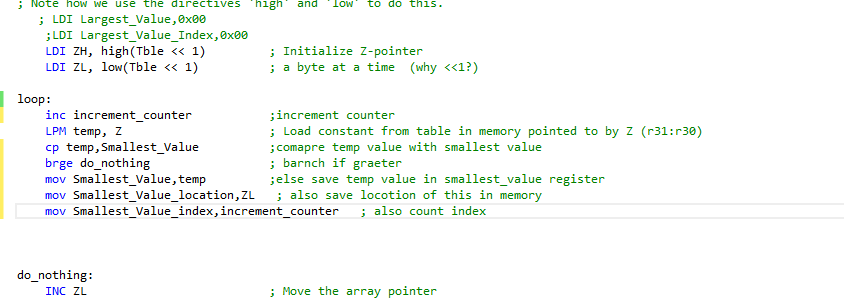
**Processor status**

**Program Memory** 

The above memory and the registers shows that the maximum value is 7B which is equal to 123 in decimal and also it is stored at 5C location.

**Design implementation for finding Smallest number in array:**

The main part of the code is pasted below.



In the starting of the loop, the counter is incremented in start to show the number of iterations and the index of number at which the largest number is present. Then the number is loaded from memory pointed by Z register and that value is stored in temp variable. After that this value in temp is compared with the value already present in the Smallest\_value. If temp value is smaller than smallest\_value then update smallestest\_value variable with temp value. Also the location and index of that biggest value is saved in smallest\_value\_­location and in smallest\_value\_index via ZL and counter variable. In the end we increment the ZL and next iteration occurs.

In this part the smallest number is to be find. It means the the more trhe 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 examp 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

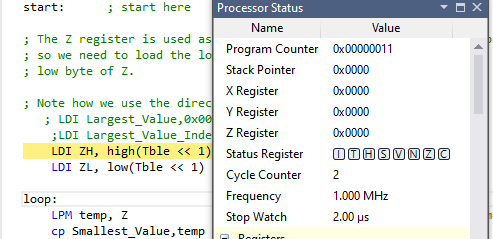
+ 1

1001 0111 (0x97)

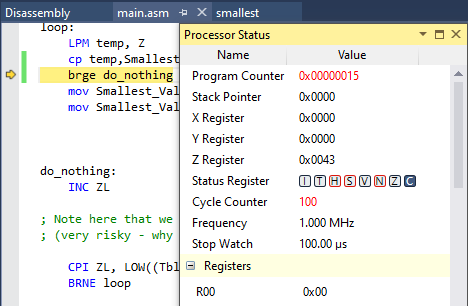
Hence in output we will see -105 as 0x97

**Example When temp value> greatest\_Value:**

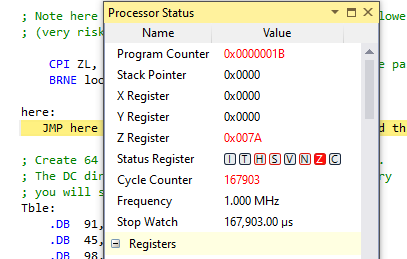
**Status/Flag Regsister:**

****

**Initial Condition**

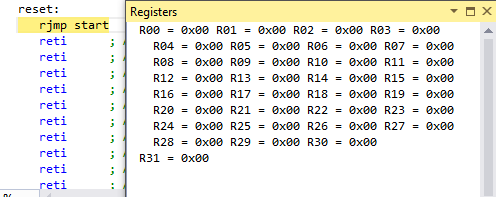
****

**During**

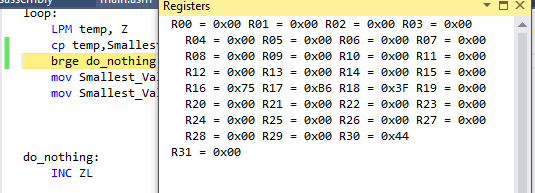


**After**

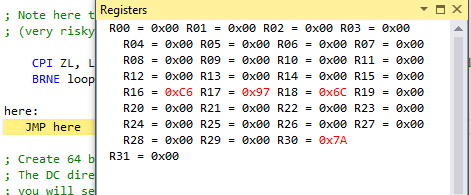
**General Purpose Registers:**

****

**Initial**

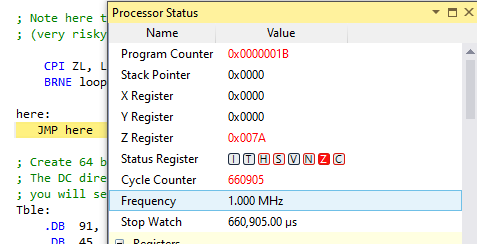
****

**During**

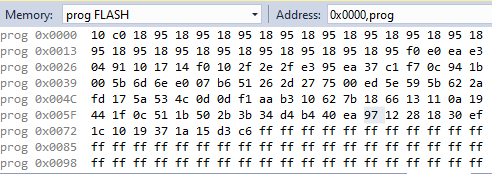
****

**After**

**Final Result when program jumps to here:**

****

**Processor status**

**Program Memory** 

The above memory and the registers shows that the minimum value is 0x97 which is equal to -105 in 2’s complement form and also it is stored at 0x70 location.

**Conclusion:**

We have learned how to implement the linear search algorithm to find the largest and the smallest number in an array. Although there are also many approaches which can be used to located number in array. In this lab we have also learned how to using assembler and debugger in Atmel studio and got to know that how registers stores and process data. By using more branch and test instructions we can make this program more efficient and flawless. The same algorithm can be applied on any avr microcontroller like atmega8, atmega16, atmega32 etc. These code gives best insights of registers, memory and about CPU status when any program runs. The key learning points were branch instructions , LPM instruction, Load instructions and compare instructions.

Reference:

1. Microchip, Atmel 8-bit AVRMCUFLASH Microcontroller, "Atmel-8155-8-bit -Microcontroller-AVR-ATmega32A\_Datasheet"
2. Microchip, Atmel 8-bit AVRMCUFLASH 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 AVRMCUFLASH 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,