

**INTRODUCTION TO ASSEMBLER USING**

**MICROCHIP STUDIO 7.0**

**Lab#01**

**Student name:**

**Student Number:**

**Lecturer name:**

**Lab instructor:**

**Subject name:**

**Introduction:**

In this lab we have to learn how to use Microchip studio 7.0 assembler and debugger by the help of a small program. This program is about the linear search in an array which we mostly use in High level languages. The linear search algorithm is used in 2 ways. One is to find the largest number in an array of signed and unsigned numbers. Other one is to find the smallest number in the same array. These two tasks will be done in assembly language and the output and insights of registers will be observed in microchip studio simulator. The signed number may be any between -128 to 127. And atmega32 controller is used for this.

**Background:**

This lab uses the linear search method to find the largest or smallest number in an array. The working principle of this method is that it checks each number in an array and compares it with the largest number or smallest number which is already present. The advantage of this algorithm is that it can find our desired element from both sorted and unsorted array. On the other hand its disadvantage is that it checks each number and compares. If the length if array is in thousands than thousand iterations are done which makes our program slower ,less efficient and time consuming for the processor also. The other methods like binary search, merge sort are also available. Their advantage over linear search is that they are fast and can search our desired number in very less iterations as compared to linear search. But they have disadvantages too. Because for these algorithms the array should be sorted in a descending or ascending order. In this we will try to implement linear search in assembly language.

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

for i = 0 to 63 do{

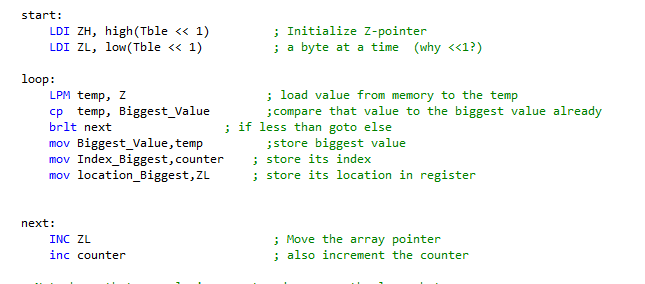
if table[i] > largest\_so\_far the{  
 largest\_value\_so\_far = table[i];  
 largest\_value \_index = i;  
 }  
}

The above pseudo code is the implementation of linear search algorithm in C language. The outer loop contains i as counter variable and it start from 0 and ends when the value of i reaches at 63. In this loop if statement is used and this if statement compares the recent value with the value which is greater till now. If the condition of if fulfilled then the largest value is updated and the index of that largest value is also saved in a variable. If condition does not fulfill the only value of loop counter i incremented and starts the program start new iteration.

**Discussion:**

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

The main part of the code is pasted below.



At the beginning of the loop the Z points to the memory (first number of the Array) and LDI instruction loads that value and stores that number in the temp register. Now that value is compared with the previous large value by CP instruction. As a result of this comparison the value of S changes. If S=1 then it means that the new value is less than previous old value. And at this point it moves to the next position where it increments the ZL register as well as counter. While if new value in temp is greater than the value present in Biggest value variable then the biggest value updated and also the index and location of that value is also saved.

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

LPM instruction is used to get value from that location which is pointed by Z and then this instruction copies that number in temp. And here temp represents the R16 register. Z is 16 bit register and it means that it can address 64k memory and in other word it can access 65536 locations. Z register consists of two parts which are ZL and ZH and each part is of 8 bit in length.

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

The Z register as we have discussed contains two parts ZL and ZH. So to increment the Z register we increment the ZL register by 1 or 2 or which increment we want. After incrementing in the ZL the Z register start pointing the next number in our array. And ZL is incremented 64 times in our lab.

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

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

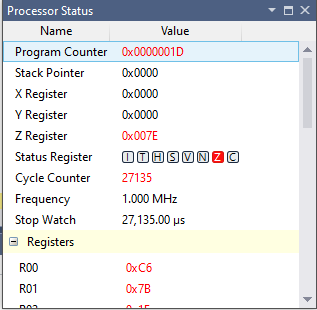
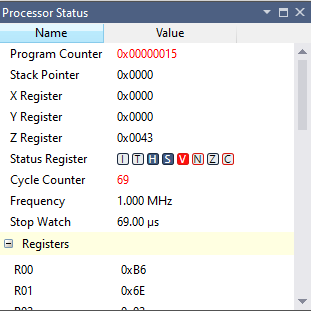
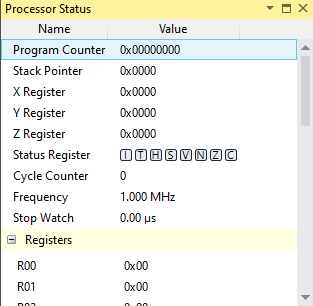
The end of table is detected via the instruction written above. The above instruction is CPI means that it compares the register value with any immediate number. Here CP compares the ZL register with the Low byte of table and checks either ZL reaches to 64th position of the array or not. If it reaches 64 then the BRNE instruction becomes wrong and program will move to here position and remain there forever.

In this lab we have also used the **BRLT** which is used for only signed numbers. In case of unsigned numbers we use the **BRLO** instruction. Both instructions will fall through only when the left hand operand is less than right hand operand. Both instructions take decision on the bases of S flag. If S=1 the BRLT instruction works. And before this instruction we have also used the **CP** instruction compares two operands and updates the flag accordingly. Here the CP instruction is used as compared to the Sub instruction because the CP instruction compares the both operands and updates flag without changing the values in those registers. But by sub instruction the destination operand’s value has been changed. Hence the CP is preferable on the Sub instruction. In the end the program jumps at the here point infinitely after 64 iterations as we have to search only that array.

**Example When temp value> greatest Value:**

**Status/Flag Register:**

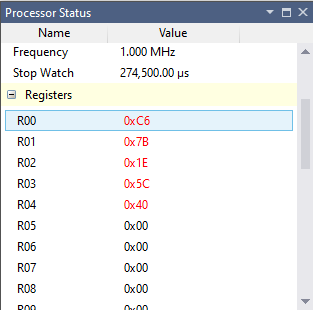
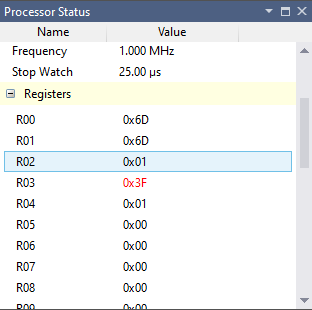
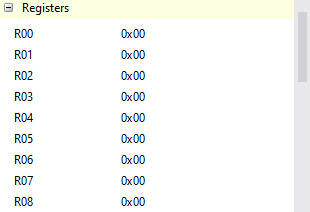
The status of the flags is shown at the start of programe, during the program and end of program.

****

**Initial During End**

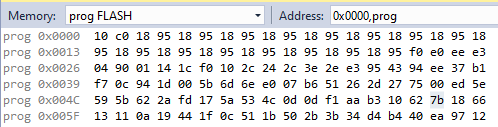
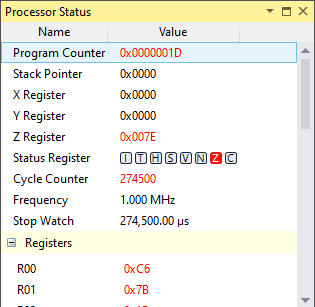
**General Purpose Registers:**

The status of the general purpose registers is shown at the start of programe, during the program and end of program.

****

**Initial During End**

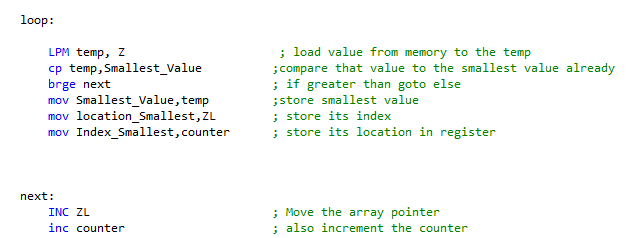
**Final Result when program jumps to here:**

****

**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. Here the temp value is stored in R0 register while the Largest value is stored in R1.index of that value is stored in R2 and the location of that number in memory is stored in R3. And the loop counter in saved in R4.

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

The main code is listed below:

The min part of this program is shown in above pic. The temp variable contains the value pointed by the Z register. This value came from the array of the numbers. After that the value is compared with the already presented value in smallest register and if temp value is greater than the smallest value then it moves to the next label where the ZL and counter is incremented. After that the program checks either it was 63th number from the array. If it was 63th then the program terminates else it again goes to starting point. After loading the next value in temp via LPM this temp instruction is compared with the smallest value again, if it is low then we update the smallest value array. Also save the location of the element present in memory and its index.

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 -93 so

105=0101 1101(0x5D)

Taking 1’s complement becomes

1010 0010(0xA2)

Adding 1 for taking 2’s complement

1010 0010

+ 1

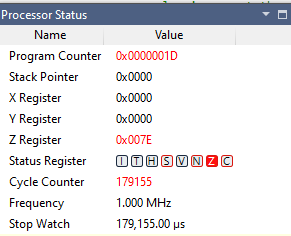
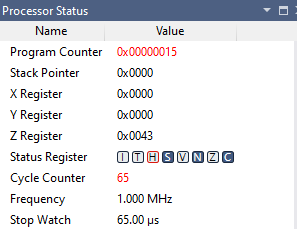
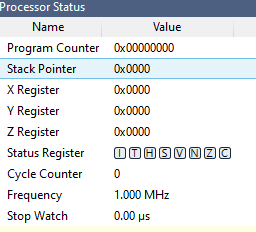
1010 0011(0xA3)

Hence in output we will see -93 as 0xA3

**Example When temp < Smaller Value:**

**Status/Flag Register:**

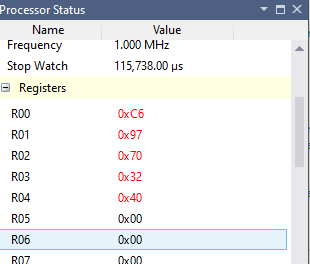
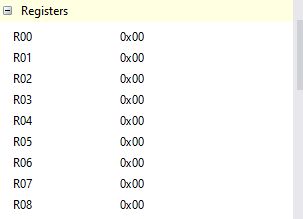
The status of the flags is shown at the start of programe, during the program and end of program.

****

**Initial During End**

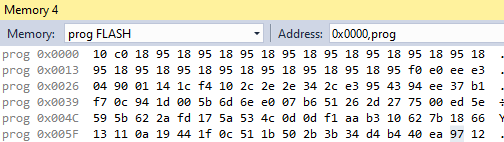
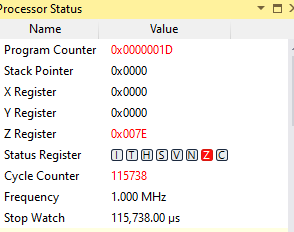
**General Purpose Registers:**

The status of the general purpose registers is shown at the start of programe, during the program and end of program.

****

**Initial During End**

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

****

**Processor status Program Memor**

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:**

To conclude all, in this lab we have learned the usage of assembly program for atmega32 and also we have learned the how to use assembler and debugger in microchip studio. We have also learned that how we can implement the high- level algorithms like binary search, linear search, merger sort and other methods in assembly language. Although this program works well but it can be made more optimized and time efficient by using some more branch instructions. We have also taken the insights of the registers and flags which has been changed accordingly during program. The both programs works were related to the searching algorithm and working behavior of both the programs was seen almost identical to each other.

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,