## CS281: Lab2 – Working with Arrays

**Introduction**

For this lab we will be digging into how to do some basic array processing with loops. You will be provided with some code that sums up all elements in an array to study. This code will provide the basis for the second part of the lab where you will write your own program to compute the minimum and maximum value in a provided array.

**PART 1: Understanding the Provided Program that Sums the Array Elements**

Walk through the provided code with the debugger and make sure you understand what is going on. The hard coded values for the array named A contain the numbers from 1 to 10. After the program is executed, it should print out: “The sum of the array is: 55”

Here is the code for part 1:

```
.data

    A : .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
    sum : .word 0
    sz_A : .word 10


    prompt:   .asciiz "The sum of the array is: "
    new_line: .asciiz "\n"

.text
    .globl  main      # assembly directive that makes the symbol main
                      # global and this is where execution starts

main:
    la s0, A        # s0 = &A[0]
    lw s1, sz_A     # s1 = sz_A
    li t0, 0        # t0 = 0  i will be the index
    # we will use t1 to store the current array element
    li t2, 0        # t2 = 0  sum will be stored here

    #for each array element we will be first calculating the
    #address using A[i] = &A + (i * 4)

sum_loop:
    lw t1, 0(s0)
    add t2, t2, t1
    addi t0, t0, 1
    addi s0, s0, 4
    bne t0, s1, sum_loop
```

```
#now save the total in sum variable which is in t2
    la t0, sum
    sw t2, 0(t0)


#print the results
    #print the prompt
    li a0, 4        # 4 is syscall for print_str
    la a1, prompt
    ecall

    # Print the sum value
    la t1, sum
    lw t1, 0(t1)
    li a0, 1        # 1 is syscall for print_int
    mv a1, t1
    ecall

    #print the newline
    li a0, 4        # 4 is syscall for print_str
    la a1, new_line
    ecall

    #now exit
    li a0, 10         # Exit code for ecall
    ecall
```

**Deliverables for Part 1:**

1. Walk through the sample code in the debugger until you understand it. Modify the input arguments and make sure the program still operates correctly. (nothing to hand in).
2. To illustrate you can follow the code, comment the code in the **`sum_loop`** to demonstrate your understanding. For this part just hand in the code for the **`sum_loop`** with your comments and not the entire program.
3. Extra credit (+10 points). Recall that the **`.data`** assembler directive controls where memory should be allocated (and initialized) for static items used by your program. Also note that **`A`**, **`sum`**, and **`sz_A`** are nothing special, they just alias the memory location where this value is stored in memory. For example, the instruction **`la s0,A`** does nothing more than load the memory address of **`A`** into **`s0`** register.

   With the above being stated, lets say we deleted the line **`sz_A: .word 10`** given we have the ability to calculate the size of **`A`** directly noticing that the **`sum`** variable is located directly after the **`A`** array. Thus, we can obtain the **total number of bytes**

used to store **A** by calculating `sz_A_in_bytes = &sum - &A`. However, the total number of bytes is not what we want, we need the total number of words to control the loop. Recall that each word is 4 bytes, so we need sz_A_in_words = sz_A_in_bytes / 4. We have not covered division yet, and it is a very slow operation that should be avoided unless absolutely necessary. Fortunately, since dividing by 4 is a power of 2, there is a very fast way we can do this using bit shifting instead of addition. Hint, it will involve using the **`srli`** instruction.

What to do and hand in:

a. Copy the provided sum program as a starting point
b. Delete the line of code in the .data segment – **`sz_A: .word 10`**
c. Replace the line of code **`lw s1,sz_A`** with some new code (it will be more than one line) that stores in the **`s1`** register the total number of words in the **A** array.
d. Comment the new code that you created explaining what you did.

**PART 2: Writing your own Array Processing Program**

For this part of the lab you will be writing your own array processing program to find both the minimum and maximum values in an array. Start by creating a new file called **`minmax.s`** and copy all of the code below to get you started:

```
.data

    A : .word 5, 4, 3, 2, 1, 10, 9, 8, 7, 6
    sz_A : .word 10
    min : .word 0x7FFFFFFF      #max signed value 2^31 – 1
    max : .word 0x80000000      #min signed value –2^31

    prompt_min:   .asciiz "The minimum value in the array is: "
    prompt_max:   .asciiz "The maximum value in the array is: "
    new_line:     .asciiz "\n"

.text
    .globl  main      # assembly directive that makes the symbol main
                      # global and this is where execution starts

main:
    # START YOUR IMPLEMENTATION OF MIN_MAX HERE
```

The goal of this program is to traverse the array, keeping track of the minimum and maximum value in this array. At the end of processing the loop your program should produce the following output given the starter code provided above:

The minimum value in the array is: 1
The maximum value in the array is: 10

The pseudo code for this program is as follows:

```
s2 = min   //note that min from .data is initially the largest
           //possible signed integer possible in 32 bits
s3 =max    //note that max from .data is initially the smallest
           //possible signed integer possible in 32 bits

i = 0  //initial index of the first element in the A array
loop:
 v = A[i]  //get the value of A[i]
 if v < s2:
  s2 = v
 if v > s3:
  s3 = v

 i++
 if (i != sz_A):
  goto loop:

//print results
```

Feel free to harvest the code from the sum program to control looping over the array and printing the results.

**Deliverables for Part 2:**

Please hand in commented source code showing your implementation showing that you can correctly identify the minimum and maximum values in an array (after you have tested it of course). Your implantation should also print out your results for both the min and max values.