CSC 3210 Computer Organization and programming

LAB 10

Arrays

PART-1

- Arrays are common in programs. They store a series of values instead of a single (scalar) value.
- The code below defines an array in the data section, then the code prints each value out.
- This is the data section.

```
.data       # Data section, initialized variables
myarr:      .word 10, 20, 30, 40, 50, 60
myarr_end: # This will be next word after myarr
temp:       .word 0
```

- This is the code section.

```
# This is written for VSCode/Venus
.text
main:
    # Note: we use s0 to make sure the value does not change
    # when we do a call/ecall
    la   s0, myarr              # s0 points to myarr
myloop:
    lw   t1,  0(s0)             # Get the value  t1 = myarr[s0]
    # Now print the array value out
    mv     a1, t1
    li     a0, 1      # print an integer
    ecall
```

PART-1

```
# print space
        li    a0, 11      # print a character
        li    a1, 32      # space (the character to print)
        ecall
        # Update s0 to point to next word
        addi   s0, s0, 4
        la     t1, myarr_end        # Does s0 == address of word beyond myarr?
        blt    s0, t1, myloop       # if less, jump to myloop
        # print NL
        li    a0, 11      # print a character
        li    a1, 10      # NL (the character to print)
        ecall
        # exit the program
        li     a0, 17
        li     a1, 0               # 0 for everything is OK
        ecall
```

- The above code prints each array value out, in order, with a space in between. Call this something like "array_print.S", and verify that it works.

PART-1

The code below prints out the sum, one partial result at a time. That is, it prints the sum that it has so far, so the first time it will be the first array value, then the first plus second array value, then the sum of the first three, etc.

```
# This is written for VSCode/Venus
.text
main:
    # Note: we use s0 to make sure the value does not change
    # when we do a call/ecall
    la   s0, myarr            # s0 points to myarr
myloop:
    lw   t1,  0(s0)           # Get the value  t1 = myarr[s0]
    la   t2,  temp            # Get the value  t2 = address of temp
    lw   t3,  0(t2)           # Get the value  t3 = temp
    add  t3,  t3,  t1         # t3 = temp + myarr[s0]
    sw   t3,  0(t2)           # store t3 in temp
    # Now print the sum value out
    mv     a1, t3
    li     a0, 1      # print an integer
    ecall
    # print space
    li    a0, 11     # print a character
    li    a1, 32     # space (the character to print)
     ecall
```

PART-1

```
# Update s0 to point to next word
        addi   s0, s0, 4
        la     t1, myarr_end        # Does s0 == address of word beyond myarr?
        blt    s0, t1, myloop       # if less, jump to myloop
        # print NL
        li    a0, 11     # print a character
        li    a1, 10     # NL (the character to print)
        ecall
        # exit the program
        li     a0, 17
        li     a1, 0              # 0 for everything is OK
        ecall
    .data       # Data section, initialized variables
    myarr:     .word 10, 20, 30, 40, 50, 60
    myarr_end: # This will be next word after myarr
    temp:      .word 0
```

- Verify that this works.
- Next, modify the code to only print the final sum value, but before doing this, make a copy of the code first.

PART-1

QUESTIONS:

1. **How do you know if the series of values is correct?**
2. **How is the address of "myarr_end" different from the address of "temp"?**
3. **Before the lw t1, 0(s0) command, which registers, i.e., s0, t0, t1, must be set to a value? If they are not, what will happen? (You can simply explain this.)**
4. **Imagine that a fellow student asks you the following, how would you explain the reason? After the instruction la t2, temp, we have lw t3, 0(t2). Why do we need this second instruction if the t2 already has "temp"?**
5. **The addi s0, s0, 4 is done intentionally. Why is the "4" there instead of "1"? What happens if the command is addi s0, s0, 1 instead? (Show this as a program, and explain its output.)**
6. **Notice that the loop has the length of 6, and that the "blt" instruction compares s0 to the address after the array. We would do this differently by counting up to 6, that is, counting the number of elements in the array instead of the address where the array ends. However, "hard-coding" the array length is a bad idea. How could we use "define/equ/eqv" to code the array length in a better way?**

PART-2

- The objective for this lab is to add to the code, to determine the minimum value, the maximum value, and the average value. You will need to define storage for each of these in the .data section (or the .bss section).
- Keep the part that prints each value in the array, only alter it to print the values on a single line. Also, have the program print something before the values, like "The array has the following values:".
- In the program, the first thing to do is to initialize the minimum and maximum values. Some programmers try to use values that are clearly wrong for initialization, such as 100,000 for the minimum and 0 for the maximum. Then, looking at each value, replace the minimum and maximum with anything that is smaller or larger, respectively.

Questions:

- **Suppose that the array has values 90, 120, 10. Describe (in a step-by-step manner) how a program would examine each one, and update the minimum and/or maximum. When finished, would we have the correct minimum and maximum?**
- **Why are the values 0 and 100,000 not approproiate? Show an example list of numbers that would make these initial values wrong for the minimum, then another example list for the maximum.**

PART-3

- The idea of looking at each value in turn and updating the minimum and/or maximum is good. However, the initialization should be with a value in the array. To do this, we can copy the first array value to the minimum, and the maximum.
- Have the program print some text, such as "The minimum is:" then the minimum value. Likewise, it should print text followed by the maximum value. And it should print some appropriate text followed by the average.

Questions:

1. **Suppose that the array has values 90, 120, 10. Initialize the minimum and maximum to the first value. Then describe (in a step-by-step manner) how a program would examine each one, and update the minimum and/or maximum. When finished, would we have the correct minimum and maximum?**

PART-4

- Now code this algorithm into the assembly language program. It should initialize the minimum and maximum with the first value of the array, then examine each value in the array (starting with the second value), and update the minimum and maximum as needed.

Questions:

1. **Suppose that the array has only the value 90. How should you alter the program to make it work with only one value?**
2. **Suppose that the array is empty, that there are no values in it. How should you alter the program to make it still work?**

PART-5

- To find the average, we can make a sum of the values. Then divide the sum by the number of values in the array. Store this average in another memory location.

Questions:

1. **The average of 10, 11 is reported as 5. Why is it 5, and not 5.5?**
2. **Would the average still work if some of the values are negative? Why or why not?**
3. **Would the average still work if all of the values are negative? Why or why not?**
4. **If the array is empty, what is a reasonable output?**
5. **Suppose that the array is very long, such as thousands of values. Would the minimum, maximum, and average code still work? Why or why not?**
6. **Given code that prints an array, it sounds trivial to add code to also find and print the minimum, maximum, and average. How does the original array printing code compare to the final code? Is this what you expected, and why or why not?**

- Include comments, and meaningful identifiers. Make sure to show the final version of the assembly language program, and the test run(s).

IMPORTANT NOTE:

Remember that we will grade your lab report so it is vital to turn that in. The other files (your code, a text version of any log file, etc.) are to document your work in case we need more information.