Lab Exercises 1 – Basic Arithmetic, Addressing and Output

(50 points)

This lab is in two parts. The first consists of a series of coding exercises to familiarize you with the use of the instructions you are learning. The second contains a series of short answer questions which you must complete for full credit. You will submit files containing job output for the exercises in Part A and the answers to Part B separately. For simplicity, a basic code skeleton has been provided for you containing storage declarations for you to use where appropriate.

Before and after you begin working on the coding exercises, be absolutely certain that you have *thoroughly* reviewed the requirements set forth in the "Coding and Documentation Standards" section on page 3 of this document. Failure to follow them will result in substantial loss of credit on this (and every following) assignment.

Part A: Coding Exercises (35 points)

To begin, upload the provided .jcl file from your local machine to Marist using the IDz Remote Systems panel by dragging and dropping from your local machine to your ASSIGNS dataset. Be sure to perform the following exercises in order, as some will require that you have completed others to work. *Leave the code from previous exercises unchanged* in your source code as you work; you will need the output for all exercises for the *single* job output file you will submit to Blackboard. In other words, as you complete exercises you should add completely new lines to your source code and avoid changing existing lines.

1. **(5 points).** Pick three registers into which to load the constant integer values 5, 10 and 200. Write the instructions to implement the equation '$200 - 10 + 5$', then `XDUMP` the contents of the registers.
2. **(5 points).** Following what you did in exercise 1, implement the equation '$15 - 87 + 9$', but this time use only *two* (2) registers while performing the subtraction and addition operations. `XDUMP` the registers again. **Note:** You are free to reuse the registers from exercise 1.
3. **(5 points).** Declare a fullword of storage and give it the label `RESA`. Use `RESA` to store the value computed in exercise 2, then `XDUMP` the fullword of program storage at `RESA`.
4. **(5 points).** Load the constant integer values 4 and 5 into registers, then implement the equation '$2*5 + 3*4$'. Since you don't know the instructions for performing multiplication, you will need to come up with an alternate method to compute a product. Declare a fullword of storage, give it the label `RESB` and use it to store the result, and then `XDUMP` the fullword of storage at `RESB`. **Note:** You are free once again to reuse any registers used in the previous exercises.
5. **(5 points).** If necessary, write the instructions to reload the values stored at `RESA` and `RESB` into registers. Write the instructions to convert and store them in character format at `ORESA` and `ORESB`, respectively, then print all 133 bytes of the print line starting at `PRNTLINE` in the job output.
6. **(5 points).** Copy the values stored at `RESA` and `RESB` into the first two elements of the three-fullword sequence (array) starting at `SEQA`. They will probably both already be sitting in registers, but if not then you'll need to load them back in again before storing them out. `XDUMP` the 12 bytes of program storage containing the sequence to the job output.

7. **(5 points).** Declare a new three-fullword sequence `SEQB` using the same method by which `SEQA` is declared. Write the instructions to copy the three-fullword sequence of values from `SEQA` to `SEQB`, then `XDUMP` the 12 bytes of storage starting at `SEQB`.

Part B: Short Answer (15 points)

Write your reponses to these short essay questions in a separate document, using MS Word or a similar program. Each response should be the equivalent of a short paragraph, about 3 or 4 sentences long on average.

1. **(5 points).** List the two major uses of the `LA` instruction mentioned in class. How does it differ from the `L` instruction?
2. **(5 points).** The `DS` and `DC` instructions have very different effects on how program storage is initialized. Why would using a `DS` when you should have used a `DC` fail to work as expected?
3. **(5 points).** In Assembler, there are no built-in variable constructs as we're accustomed to in high-level languages, however the concept of a variable is still extremely useful when thinking about how we write programs. In a few sentences, describe how you would set up and use program storage to act as a variable, e.g. in the computation of the result of an equation such as $a + b$.

Coding and Documentation Standards

Assuring that your code follows proper coding conventions and documentation standards is a critical habit in the workplace, and can make the difference between keeping or losing your job. Your work will therefore be graded, in part, on your ability to adhere to the following coding and documentation standards:

Coding standards

1. **Available registers.** You may only use registers 2 through 11 (inclusive) for loading values, addresses and performing math operations. Registers 0–1 and 12–15 are reserved for special use.
2. **Loading constant values.** Unless otherwise instructed, when you are told to load a *constant* integer value into a register you must use the `LA` instruction. You may *not* declare the value in program storage and then load it with `L`.
3. **Creating arrays.** When told to declare an array to store a sequence of values in program storage, you must declare it using a *single* `DS` or `DC` instruction prefixed with a label.
4. **Accessing arrays.** For this assignment, when you are told to access elements (locations) in the array for loading or storing values, you must first load the array's base address into a register using `LA` and then use that register with an appropriate displacement in a relative address as the second operand of your `L` or `ST` instruction.

Documentation standards

1. **Documentation box.** Your program code must contain a standard documentation box (or docbox) immediately following the opening JCL statements:

```
*****************************************************************
* CSCI 360-2            LAB EXERCISE n              SPRING 2022 *
*                                                               *
* NAME: (Your name)                                             *
* DATE: (Date of submission)                                    *
*                                                               *
* (Brief Description)                                           *
*                                                               *
* REGISTER USAGE:                                               *
*   Ra - (Usage description)                                    *
*   Rb - (Usage description)                                    *
*   ...                                                         *
*                                                               *
*****************************************************************
```

   n – Number of the current lab exercise (i.e., 1)
   a – Lowest register number
   b – Next register number

2. **Inline documentation.** Roughly 80% of your lines of code should have comments separated from and following the operands of the Assembler instructions.

If your submitted code violates *any* of these standards, then you will automatically lose 50% of the points ***you would have received***, ***AFTER*** your program's functionality and output are assessed.