Lab 3 – Tables and DSECTs

(40 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 both storage declarations for you to use where appropriate and in-stream program data.

Before you submit your job output for the coding exercises (which will include the source code), be absolutely certain that you have *thoroughly* reviewed the requirements set forth in the "Coding and Documentation Standards" section on pages 2 & 3 of this document. Failure to follow them will result in substantial loss of credit on this assignment.

Part A: Coding Exercises (30 points)

1. **10 points.** In this exercise you will read the first table from the in-stream program data (there are two) into the `FTABL1` table provided by the JCL skeleton. All values should be stored as fullwords. Be sure to stop at the '*' in the first byte of the following record (you don't need to do anything besides check for it, so it can be safely ignored afterwards). Use the first method discussed in class—i.e., using the displacement (D) of the D(X,B)—in both the `XDECI` and `ST` instructions. `XDUMP` the table after you've filled it in. Note that you won't be able to use an inner loop to handle the columns, since displacements are hard-coded.

2. **10 points.** Next, read in the second table from the in-stream program data and store it using the second method discussed in class, i.e., using an index register to dynamically apply the column offsets. Store all values as fullwords. You will need to declare a second table in program storage large enough to accommodate the data. Once finished `XDUMP` the table. Note that you'll need to implement an inner loop to handle reading and writing columns from the record (the data is in space-separated column format) to the table.

3. **10 points.** Create a `DSECT` for the columns in the second table (declared in Exercise 2) in program storage. You may give the `DSECT` and `DSECT` field labels any name you choose, but as per convention they must all start with '$'. Use the `DSECT` labels to load the data in each row, and put them into a print line buffer—*which you will also need to declare*—with `XDECO` and then send the print line buffer to job output. You may optionally declare a second `DSECT` to associate with the print line buffer for the `XDECO`.

Part B: Short Answer (10 points)

1. **5 points.** A program needs to read and store multiple rows of data into a table declared in storage. Assume that there *could* be more rows of data read from file than there is room to accommodate in the table. Briefly describe a way to prevent your program from writing past the end of the table. ***Note:*** There are several ways to do it, but you only need to talk about one.

2. **5 points.** Suppose that you have a table declared in storage as

   ```
   TABLE      DS     CL340
   ```

   Assume that 1) Each row stores 10-bytes of character data in the first column; 2) the table is exactly large enough to accommodate 10 rows; and 3) besides the first column, all other columns store fullwords. Write a `DSECT` (the labels can be anything, though make sure to follow the standard `DSECT` label naming convention) that can be used to access the columns in each row. ***Hint:*** You'll need to figure out how many columns there are in total first.

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. **DSECT declarations.** Your `DSECT`s must be declared at the head of your program, after the docbox and before the main `CSECT`.
3. **DSECT labels.** As discussed in class, you must prefix all labels associated with a `DSECT` with a '$'.

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.