Lab 4 – Packed Decimals

(80 points)

This lab consists of a single part, which presents a series of coding exercises to familiarize you with the use of the instructions you are learning. You will submit a file containing job output for the exercises. 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 page 2 of this document. Failure to follow them will result in substantial loss of credit on this assignment.

Coding Exercises (80 points)

1. **5 points.** Read in the first record from input, and write the code to pack the single EBCDIC numeric representation into a packed decimal field called PMULT. Use only the minimum number of bytes required to store it in packed decimal format. `XDUMP` the storage containing those fields.

2. **5 points.** Read in the next record from input. Pack and add the values in the record together, then `XDUMP` the storage containing the total.

3. **5 points.** Declare a temporary packed decimal field in storage called PCALCTMP, and use it to perform multiplication between the result in exercise 2 and PMULT. Be sure to properly prepare the first operand to the `MP` instruction.

4. **5 points.** Assume that the two packed decimal numbers just multiplied together, the sum and PMULT, had 3 implied decimal places and 2 implied decimal places, respectively. Round the result to 1 implied decimal place and `XDUMP` it.

5. **5 points.** Divide the result obtained from exercise 4 by the value 2.25, provided as a literal in the `PD` instruction, and then use shifting to give the result 2 implied decimal places. Don't forget to prepare for the division as demonstrated in class. `XDUMP` the result.

6. **5 points.** Declare storage for a print buffer, then use the `ED` instruction (with an appropriate mask) to convert the result from exercise 5 to printable format. Make sure that the mask uses a space as the fill character, includes the proper number of digit selectors, turns on significance in the correct spot (as demonstrated in class), and includes a decimal message character in the correct position for a packed decimal value with two implied decimal places. Send the print buffer to the job output.

7. **5 points.** Do the same as in exercise 6, this time using `EDMK` to prefix the printed value with a dollar sign ('$'), as demonstrated in class.

8. **15 points.** Declare an array of packed decimals called PCOEFFS to store the sequence of values read from the next record, and store them. Next, declare a table to store the sequences of values read in from all remaining records. Pack and store the values in each record as a new row of the table. `XDUMP` both the array and the table.

9. **20 points.** Iterate through the rows of the table. At each row, multiply the value in each column by the value in the corresponding column of the PCOEFFS array and add the results together, then divide the summed result by 10.12. Assume that all PCOEFFS values have one implied decimal place. `XDUMP` the final result. **Note:** There should be a separate `XDUMP` for each row in the table).

10. **10 points.** Re-implement the code you wrote in exercise 9—create a new section of code; don't change the code you've already written—using the `ED` instruction and a print buffer in place of the `XDUMP` to show the result for each row.

Coding and Documentation Standards

Be certain to follow all coding standards already established in previous assignments. Most importantly (for this assignment), *do not let lengths of packed decimal fields default!*

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.