**CSCI 360-1 Assignment 8 – Packed Decimal Spring 2019**

**150 points**

This assignment is an introduction to working with packed decimal data and instructions. It is not meant to be difficult but rather help you understand the radically different instructions and how packed decimal and zoned decimal numbers are stored and manipulated. Refer to the Lecture Notes on Packed Decimal on Blackboard to help you.

To start, copy your ASSIGN7 PDSE member and name it ASSIGN8 (Don't forget to change the END statement too!). You can also refer to the sample program provided alongside this document on Blackboard.

Replace the beginning CSECT and USING statements with the following standard entry linkage (more about this later):

ASSIGN8 CSECT

PRINT NOGEN DO NOT EXPAND MACROS

STM 14,12,12(13) SAVE REGS IN CALLER'S SAVE AREA

LR 12,15 COPY CSECT ADDR INTO R12

USING ASSIGN8,12 ESTABLISH R12 AS THE BASE REG

LA 14,SAVEAREA R14 POINTS TO THIS CSECT's SAVE AREA

ST 14,8(,13) STORE ADDR OF THIS CSECT's SAVE AREA

ST 13,4(,14) STORE ADDR OF CALLER'S SAVE AREA

LR 13,14 POINT R13 AT THIS CSECT'S SAVE AREA

Replace the final BR 14 statement in your program with the following standard exit linkage (more about this later):

SR 15,15 R15 = RETURN CODE OF 0

L 13,4(,13) POINT R13 TO CALLER'S SAVE AREA

L 14,12(,13) RESTORE REGISTER 14

LM 0,12,20(13) RESTORE R0 THRU R12

BR 14 RETURN TO CALLER

At the top of your storage area following the LTORG, declare the following:

SAVEAREA DC 18F'-1' REGISTER SAVE AREA

Like every Assembler program in the real world, your code for this program will be written between the standard entry linkage and the standard exit linkage. At this point, see if you can get it to completely execute. If you get the same results as you did with your Assignment 7 program, you will know you've copied the standard entry and exit linkage into your program correctly. **Note that register 12 will now be used as the base register of your entire program, not register 15.**

Note also that the standard exit linkage first zeros out register 15 and also note that, although we are restoring ALL of the other registers, we are not restoring register 15 before the BR 14 in the standard exit linkage. If we did, this would put something different in register 15 and the SR 15,15 would have been worthless. So, this is the standard way we send back a return code of 0 to the caller.

Now that you're done with the above, here we go!

This program will be very much like your Assignment 7 program but the arithmetic will all change. We are going to use packed decimal arithmetic instead of binary arithmetic. The input data set is:

DSN=KC02322.CSCI360.DATASP19(DATA8),DISP=SHR

Each record has the following format:

|  |  |  |
| --- | --- | --- |
| **Cols.** | **Field Name** | **Range** |
| 1-8 | Employee ID | 00000000 to 99999999 |
| 9-13 | Hourly Pay | $000.01 to $999.99 |
| 14-18 | Hours Worked | 000.01 to 999.99 |
| 19-23 | Deduction Amount | $000.01 to $999.99 |
| 24-28 | Bonus Amount | $000.01 to $999.99 |
| 29-53 | Employee Name | alphanumeric |

* Employee ID, Hourly Pay, Hours Worked, Deduction Amount and Bonus Amount are all zoned decimal numbers in the data set.
* Hourly Pay, Hours Worked, Deduction Amount and Bonus Amount all have two ***implied*** decimal places (dollars and cents).
* The definition of the input record could be defined in storage as follows.

RECORD DS 0H

IEMPID DS ZL8 🡪 packs into 5 bytes (8 / 2 rounded = 4 + 1 = 5 bytes)

IHRPAY DS ZL5 🡪 packs into 3 bytes

IHOURS DS ZL5 🡪 packs into 3 bytes

IDEDUCT DS ZL5 🡪 packs into 3 bytes

IBONUS DS ZL5 🡪 packs into 3 bytes

IEMPNME DS CL25

DS CL27

* Remember that, when you work with packed decimal numbers, it is your responsibility to keep track of where the decimal point is implied in the packed decimal field. Decimal points are NEVER stored in numeric storage!

A good example of where this is especially important is the result of multiplication. If you will remember from your elementary school math, when you multiply two numbers with decimal places, the product has as many decimal places as the number of decimal places of the multiplier ***added to*** the number of decimal places of the number being multiplied. For example, if you multiply a number with four decimal places by one with two, the product will have six decimal places.

Hint: This becomes important when you multiply the Hourly Pay by the Hours Worked (described next).

* You are going to be multiplying a number that can be as large as 99999F (3-byte packed hourly pay rate) by a number that can be as large as 99999F (3-byte packed hours worked). The largest result, or 99999 \* 99999, would be 9999800001F which fits in a minimum of 6 bytes with a leading 0:

09 99 98 00 00 1F 🡨 the result with four decimal places

This 6-byte field needs to be defined in storage and will hold the result, i.e., gross pay, when you're done with all of the arithmetic.

BEFORE you subtract the deduction and add the bonus, you need to round the result of the multiplication from four to two decimal places. Use SRP to shift off the two rightmost digits with rounding.

* All of the Deductions, Bonuses and Gross Pay Amounts should be accumulated and printed as shown in the exact output provided.
* Make sure that your output looks as close to the example output as possible. The TA cannot take off points if yours matches the example.
* Any program that XPRNTs a blank line will earn 0 points. Always use the carriage control character to double space your lines. Note that you will not see double-spaced lines in SDSF. Press F10 to scroll one byte to the left to see the carriage control character and verify that they are correct.
* It is recommended that you get the ID, name, hourly pay, hours worked, deductions and bonus fully formatted and printing correctly on the print line BEFORE doing any of the arithmetic and then formatting the gross pay on the print line.

**Missing Output after mar\_ftp.exe**

Those who can see output in SDSF on Marist but cannot see it or part of it after using mar\_ftp.exe to download it to their own laptop or PC to submit it must do the following to fix the problem:

1. Make sure the lengths of ALL output line definitions in storage are exactly 133.
2. Make sure each of them have a valid carriage control of a space (single spacing), a zero (double spacing) or a 1 (top-of-page).
3. Make sure that each edit patterns moved into a receiving field prior to ED or EDMK is exactly correct with exactly the right number of digit selectors.
4. Make sure that there is a DC of spaces in between each of the receiving fields defined in each of the output line definitions.

The loss of output data is caused by one or more bad (and probably hidden) characters being found in an output line.  You must count carefully and pay attention to details.

Document your program completely and submit your single .txt file on Blackboard as before.