CSCI 465 **Assignment 6: QSAM**
250 points

Overview

This assignment's objective is to give you experience using the five Assembler QSAM macros and hone your Assembler programming skills. You will create the same reports you did for Assignment 5.

Name this program `SALESRPT` and utilize the file `ASN6DATA` the same way you did in Assignment 5 (it is the same data as before).

Write an Assembler QSAM program that creates **the same** two well-designed printed reports as the COBOL program you wrote for Assignment 5. Read and follow the Programming Notes below VERY carefully.

Programming Notes (READ THESE CAREFULLY!):

- Use the QSAM Assembler program provided to you for Assignment 3 as a starting point.
- Name your `ASSIGNS` PDSE member for this assignment `ASSIGN6` and only `ASSIGN6` so that your progress can be reviewed by Mr. Decker at any point.
- Be sure to back up your `ASSIGNS` PDSE periodically; if it is corrupted or you accidentally delete it, it cannot be restored.
- Write your Assembler QSAM program using `GM` (Get Move) and `PM` (Put Move) macro formats first. Once you get it working perfectly, then change to the macro formats in the following three bullet points.
- Use the `GL` version of the `GET` macro for accessing `ASN6DATA`. Use a `DSECT` for each of the two input record types and do **NOT** move (`MVC`) the data into your program, i.e., access the data where it was located by register 1 when the `GET` with `MACRF=GL` was issued. This is considered "in situ" processing.
- Use the `PM` and `GL` versions of the `PUT` and `GET` macros, respectively, for accessing the temporary data set for low endowment amounts.
- Use the `PM` version of the `PUT` macro for writing records to the output for your two reports.
- Do not use literals except for 1) adding 1 to a packed decimal accumulator variable, i.e., page counter, or 2) any shorter numeric-edited output edit patterns. Note that longer output edit patterns should be defined in storage instead of using a literal.
- Use plenty of wisely-placed asterisks in column 1 to spread your code out vertically for easier reading.
- Macro names, `DS`s, `DC`s, mnemonics, etc., all go in column 10. Operands and storage definitions begin in column 16 (Note that the names of ***some*** IBM macros are so long that they require parameters start in a column following 16. If so, that's okay).
- Use a `PRINT NOGEN` above your Assembler program's main doc box to suppress macro expansion in your source listing.

- Immediately following the required `LTORG`, use the 32-byte boundary `ORG` technique to easily identify your storage in a dump.

- Define all of your `DSECT`s above the `CSECT`. All `DSECT` names and field names must begin with a dollar sign (`$`).

- Remember that the temporary data set will only need a **single** `DCB`.

- Test each file `OPEN` for success using `LTR 15,15`. If the return code is something other than 0, force an abend using the `ABEND` macro with a different `USER` code for each `OPEN`.

- Do **all** arithmetic in packed decimal. Only your line counter should use register arithmetic because it is never printed!

- Do **NOT** use `MVC` to move packed decimal numbers. Use `ZAP` instead **but only move them if absolutely necessary**.

- Use the `TIME` macro **ONLY** once.

- Declare `DCB`s **at the end of your storage** with any required `EODAD` EOF routines and EOF flags declared **immediately following** the `DCB` definition for that file.

- Use good line documentation and other documentation as described in the *Coding and Documentation Guidelines*.

- **Be sure that ALL 133-byte output line definitions have spaces defined in between the fields that will receive values. Any invalid or non-printable characters will result in fragmented output!**

- Put the following two lines in your code near the top of each of the two loops, perhaps immediately following your increment of the sale counter in the first loop and high-sale counter in the second, both of which should be the very first thing you do in a loop body:

```
MVI   dtlline+1,C' '
MVC   dtlline+2(131),dtlline+1
```

- Do **NOT** use subroutines of any type.

- Be sure you are using the following standard entry and exit linkage for your program:

Standard Entry Linkage

```
SALESRPT CSECT
         STM   14,12,12(13)    SAVE CALLER'S REGS
         LR    12,15           SET R12 TO R15
         USING SALESRPT,12     ESTABLISH R12 AS 1ST BASE REG
         LA    14,SAVEREGS     R14 -> CURRENT SAVE AREA
         ST    13,4(,14)       SAVE CALLER'S SAVE AREA ADDR
         ST    14,8(,14)       SAVE CURRENT SAVE AREA ADDR
         LR    13,14           R13 -> CURRENT SAVE AREA
```

Standard Exit Linkage with RC = 0 in R15

```
SR    15,15           RC = 0
L     13,4(,13)       R13 -> CALLER'S SAVE AREA
L     14,12(,13)      RESTORE R14
LM    0,12,20(13)     RESTORE R0 THROUGH R12
BR    14              RETURN TO CALLER
```

- Use the advanced Assembler techniques discussed in class. For example, to set a register to a value between 1 and 4095, use `LA`, to set a register to 0, subtract the register from itself, to decrement a register by 1, use `BCTR`, etc.

As before, what you will hand in will be one single job .txt file produced by a job with an instream Assembler program followed by a Binder step followed by a fetch and execute step similar to what you have done before.

Note that, if you were to print the reports from Assignment 5 and those produced by this assignment's Assembler QSAM program, you should be able to hold them up to the light and see only differences in the date and time. To do this, use your COBOL definitions of your print lines as a reference when defining your Assembler print lines.

You are required to use the field names provided in the document you used for the same purpose for Assignment 5. Note that Mr. Decker may alter and add to this list as he develops his own version. This will assist your TA and instructor in helping you and also form good habits.

What to Turn In

Use Retrieve Jobs to download and convert your output and submit your `.txt` file on Blackboard as before.