COMSC 260

Fall 2020

Programming Assignment 12

Worth 15 points (1.5% of your grade)

DUE: Tuesday, 12/1/20 by 11:59 P.M. on Canvas

START by downloading the 260_assign12.asm file from Canvas

NOTE: Your submission for this assignment should be a single **.asm** file and a single **.pdf** file.
The following naming convention should be used for naming your files: **firstname_lastname_260_assign12.asm and firstname_lastname_260_assign12.pdf**. The pdf file that you submit should contain the screenshots of your sample runs of the program (see below). For example, if your first name is "James" and your last name is "Smith", then your files should be named James_Smith_260_assign12.asm and James_Smith_260_assign12.pdf.

COMMENTS (worth 7.5% of your programming assignment grade): Your program should have at least **ten (10)** different detailed comments explaining the different parts of your program. Each individual comment should be, at a minimum, a short sentence explaining a particular part of your code. You should make each comment as detailed as necessary to fully explain your code. You should also number each of your comments (i.e., comment 1, comment 2, etc.). **NOTE: My comments do NOT count towards the ten comments!**

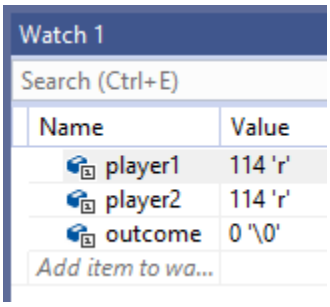
SAMPLE RUNS (worth 7.5% of your programming assignment grade): You should submit screenshots of at least **five (5)** different sample runs of your program. Your sample runs should **NOT** be the same as the sample runs that are used in this write-up for the assignment. You should also number each of your sample runs (i.e., sample run 1, sample run 2, etc.). Each sample run should show (1) the values in the player1, player2, and outcome variables at the **BEGINNING** of the program and (2) the values in the player 1, player2, and outcome variables at the **END** of the program. Some of your sample runs should use lowercase letters for the player1 and/or player2 variables to confirm that your to_upper procedure is working correctly (see below).

For example:

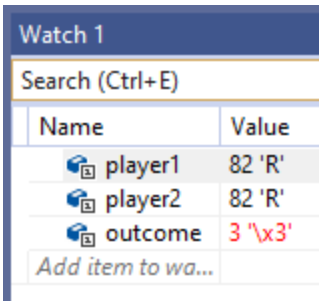
Values of player1, player2, and outcome variables at the beginning of the program:

```
.data

player1 BYTE 'r'
player2 BYTE 'r'
outcome BYTE ?
```



Values of player1, player2, and outcome variables at the end of the program:



IMPORTANT NOTE: In this class (for this assignment and ANY other assignment) **the high-level directives from section 6.7 of chapter 6 are NOT allowed to be used under any circumstances.**

This means things such as .if, .else, .repeat, .while, etc. must **NOT** be used in this assignment, or any other assignment in this class. Use of these high-level directives would make the assignments too easy, so they are **NOT** allowed for this reason.

When your program needs to make decisions, it needs to do so using the conditional jump instructions that we are learning about in class.

OK: conditional jump instructions such as je, jne, jecxz, ja jnae, jg jcxz, jp, jnp, jb, jnbe, jp, jz, js, etc. jmp (unconditional jump) is OK as well.

NOT OK: High-level directives such as .if, .while, .else, .repeat, etc.

(**do NOT use anything from section 6.7 of chapter 6 in any of your assignments!!!**)

For your twelfth programming assignment you will be implementing a game of **rock-paper-scissors** in assembly! In addition to main you will have the following two procedures:

(1) to_upper procedure

(2) RPS procedure

to_upper procedure: converts a lowercase letter to the corresponding uppercase letter (e.g., 'r' gets converted to 'R', 'p' gets converted to 'p', and 's' gets converted to 'S').

RPS procedure: Correctly stores either:

- 1 in the outcome variable by the end of the procedure (if player 1 won)
- 2 in the outcome variable by the end of the procedure (if player 2 won)
- 3 in the outcome variable by the end of the procedure (if there is a tie)

In the .data segment there are three variables (player1, player2, outcome):

```
.data

player1 BYTE 'S'
player2 BYTE 'p'
outcome BYTE ?
```

The idea is that the program will be ran with either 'R', 'P', or 'S' in the player1 and player2 variables, and at the end program the correct number (1, 2, or 3) will be stored in the outcome variable. Note that the program is also supposed to work if either of the player1 and/or player2 variables are storing a lowercase 'r', 'p', or 's' – this is why there is a to_upper procedure. By converting lowercase letters (when necessary) to the corresponding uppercase letter first, we don't have to have separate logic for handling lowercase letters vs uppercase letters in the RPS procedure. **The RPS procedure treats everything as uppercase**.

Note: the **to_upper** procedure should **ONLY** be called if the value in the player1 and/or player2 variables is a lowercase 'r', 'p', or 's'. This function should **NOT** be called if the letter is already an uppercase 'R', 'P', or 'S'. Note that the player1 and player2 variables need to be checked separately, and two different calls to the to_upper procedure may be necessary (**the two_upper procedure only converts one operand to uppercase at a time**).

In the **RPS** procedure, make sure to handle **all of the cases**:

Rock-Paper-Scissors Table		
Player 1's Choice	**Player 2's Choice**	**Outcome**
R (rock)	R (rock)	3 (tie)
R (rock)	P (paper)	2 (player 2 won)
R (rock)	S (scissors)	1 (player 1 won)
P (paper)	R (rock)	1 (player 1 won)
P (paper)	P (paper)	3 (tie)
P (paper)	S (scissors)	2 (player 2 won)
S (scissors)	R (rock)	2 (player 2 won)
S (scissors)	P (paper)	1 (player 1 won)
S (scissors)	S (scissors)	3 (tie)

NOTE: The values originally placed into the player1 and player2 variables should be preserved and not changed. The **ONLY** exception is if the register originally stores a lowercase letter, then it will be converted to the corresponding uppercase letter.

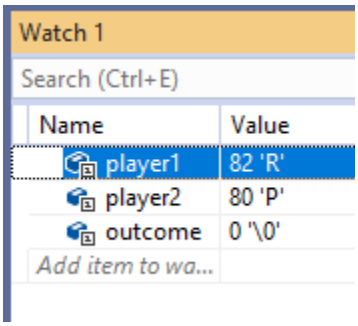
You can also assume that both the player1 and player2 variables are initialized with either 'r', 'p', 's', 'R', 'P', or 'S', so invalid input values do not have to be handled.

Sample run 1:

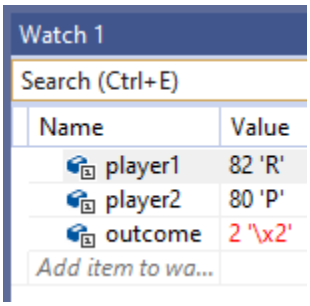
Values of player1, player2, and outcome variables at the beginning of the program:

```
.data

player1 BYTE 'R'
player2 BYTE 'P'
outcome BYTE ?
```



Values of player1, player2, and outcome variables at the end of the program:

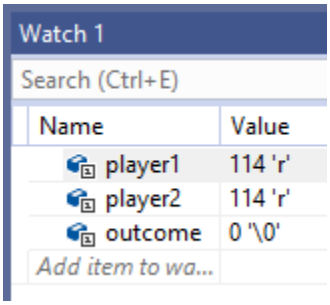


Sample run 2:

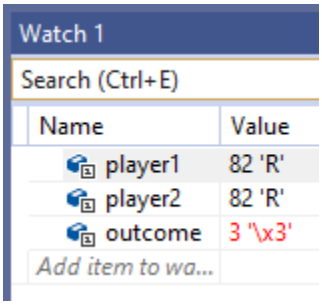
Values of player1, player2, and outcome variables at the beginning of the program:

```
.data

player1 BYTE 'r'
player2 BYTE 'r'
outcome BYTE ?
```



Values of player1, player2, and outcome variables at the end of the program:

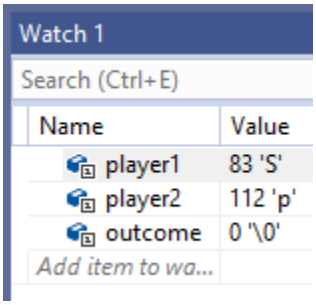


Sample run 3:

Values of player1, player2, and outcome variables at the beginning of the program:

```
.data

player1 BYTE 'S'
player2 BYTE 'p'
outcome BYTE ?
```



Values of player1, player2, and outcome variables at the end of the program: