AUBURN UNIVERSITY

CPSC 3303 Assignment

Instructions For this assignment, you must use assembly to write one program that will include five procedures. The program with the five procedures will be included in ONE file inside ONE project. These program and procedures should be implemented and tested using the *Visual Studio* IDE. For all these exercises, you cannot use built-in functions that perform the same procedures. In case of a doubt, check with your instructor.

For this assignment, you will build your own **byte** input/output **procedures** using basic **character** input/output functions provided by the author of the textbook.

This assignment differs from the previous assignment by:

1) using the procedure construct (`PROC` directive, `call` and `ret` instructions)
2) using conditional processing (`cmp`, `jb`, `jae`…). You will need conditional processing to handle **all** hexadecimal digits '0', '1', ...'9', 'A', ... 'E', and 'F'. Recall that the previous assignment limited us to use only decimal digits '0', '1', ... up to '9'.
3) Implementing your first iteration (`loop`).

Objectives of this assignment:

- to build up your programming skills in the assembly language
- to use procedures (`call` and `ret` instructions)
- to use conditional jumps (`cmp`, `jb`, `jae`, `loop`,…)
- to get familiar with *Visual Studio*
- to stress on the fact that the memory (variables) ultimately are 0s and 1s
- to use some basic character input/output functions
- to build your own basic **byte** input/output functions (using any hexadecimal digit).
- to distinguish between "numbers" and characters
-

Warning:

The executable you will build may get **blocked** by an Antivirus scanner on your system. In this case, read this from the Textbook author: "*Antivirus scanner software has improved greatly in recent years, and so have the number of viruses (one website reports 50,000 at present). Because of this, your computer's antivirus scanner may report a false positive when you build your program and refuse to let you run it. There are a few workarounds: (1) You can add your project's bin/debug folder into an exclusion list in your antivirus configuration. This is my preferred approach, and it nearly always works. (2) You can suspend your realtime antivirus scanner software, but this will leave you open to malware for a short time. If you choose this option, be sure to temporarily disconnect your computer from the Internet. (3) You can send a copy of your program's EXE file to the antivirus software vendor, labeling it as a false positive. The virus scanner I use automatically uploads any questionable EXE file to their website and either quarantines or releases the file within about 30 minutes.*" *I used the first workaround, i.e., I added my project's /debug folder to the exclusion list in my antivirus configuration.*

Input/Output Built-in Functions Allowed:

The textbook author wrote some helpful functions. For this project, you **can** use only these two built-in functions:

1) `ReadChar` : this function will allow you to read a character from the keyboard and receive its ASCII code in the register AL. This function is described in the textbook Page 165 (beginning/page).
2) `WriteChar` : this function will allow you to display on the console window (command window) a character whose ASCII code is stored in the register AL. See in the textbook Page 169 (start/page).

For this programming assignment, you must ultimately design and implement the code for the following five procedures:

1) *`DigitValue2ASCII`* : this procedure takes as a parameter the value ***v*** of a hexadecimal number stored in Register AL and returns in Register DL the corresponding ASCII. See Programming Exercise 1 for details.
2) *`ASCII2DigitValue`* : this procedure takes as a parameter the ASCII code of a hexadecimal digit stored in Register AL and returns in Register DL the corresponding value. See Programming Exercise 3 for details.
3) *`WriteHexByte`* : this procedure will allow you to display on the console window (command window) the hexadecimal number stored in the register AL. See Programming Exercise 2 for details.
4) *`ReadHexByte`* : this function will allow you to read an 8-bit hexadecimal integer from the keyboard and receive it in the register AL. See Programming Exercise 4 for details.
5) *`SumFirstN`* : this function will compute the sum ***S*** = 1 + 2 + ... + ***N*** where ***N*** is in the register AL. The sum must be stored in Register DX. See Programming Exercise 5 for details.

Note: all procedures and the main program must be included in ONE project using ONE assembly file.

Programming Exercise 1 (4 points): (Implementing *`DigitValue2ASCII`*)

The objective of this exercise is to write the **procedure *`DigitValue2ASCII`*** that converts a value ***v*** of a hexadecimal digit ***d*** into the ASCII code of the digit ***d***. The **procedure *`DigitValue2ASCII`*** uses the register AL as parameter. We assume that AL contains a value between 0 and 0Fh=$(15)_{10}$. Therefore, AL will contain the ***value*** of a hexadecimal digit '0' to 'F'. You do not have to test AL for the range.
The procedure ***`DigitValue2ASCII`*** must return in the register DL the ***ASCII code*** of the digit ***d***.

Examples:

1) If AL contains the **value** 4, procedure ***`DigitValue2ASCII`*** returns in DL the ASCII code of '4', i.e. DL ←34h .
2) If AL contains the **value** 0Ch=$(12)_{10}$, your procedure return in DL the ASCII code of 'C', i.e. DL←43h .

Programming Exercise 2 (8 points): (Implementing *WriteHexByte***)**

The objective of this exercise is to write the procedure ***WriteHexByte*** to display on the console the byte contained in the register AL: AL is the parameter of the procedure ***WriteHexByte***. This procedure must:

1) display the number stored in AL
2) display 'h' to indicate that the number is a hexadecimal number.
3) *'Display'* the 'line feed' character (ASCII code is 0Ah) to go to the next line
4) *'Display'* the 'carriage' character (ASCII code is 0Dh) to go to the next line

Displaying a 'line feed' and a 'carriage return' will produce a new line. Check what happens if you do not display 'line feed' and/or a 'carriage return'.

Examples :

1) If AL contains the number 94h, the procedure ***WriteHexByte*** must display the characters '9', '4', 'h', 'line feed', and 'carriage return'. Recall that 'line feed' is **ONE** character whose ASCII code is 0Ah and 'carriage return is **ONE** character whose ASCII code is 0Dh.
2) If AL contains the number E7h, the procedure ***WriteHexByte*** must display the characters 'E', '7', 'h', 'line feed', and 'carriage return'.

Programming Exercise 3 (4 points): (Implementing `ASCII2DigitValue`**)**

The objective of this exercise is to write a procedure that will convert the ASCII code of a digit ***d*** into the value ***v*** of ***d,*** and store ***v*** in the register **DL**. The procedure ***`ASCII2DigitValue`*** uses the register AL as a parameter. AL contains the ASCII code of a hexadecimal digit ***d*** ('0' to 'F')***:*** 30h, 31h, ...38h, 39h, 41h, 42h,.., 45h. You do not have to test AL for than range. The procedure ***`ASCII2DigitValue`*** must store in the register DL the ***value*** of the digit.

Examples:

1) If AL contains the **ASCII code** 34h (ASCII code of '4'), the procedure ***`ASCII2DigitValue`*** must store in DL the **value** of the digit '4', i.e., the value 04h. Ultimately, DL ← 04h.
2) If AL contains the **ASCII code** 44h (ASCII code of 'D'), the procedure ***`ASCII2DigitValue`*** must store in DL the **value** of the digit 'D', i.e., the value 0Dh. Ultimately, DL ← 0Dh

Programming Exercise 4 (8 points): (Implementing *ReadHexByte*)

The objective is to write the procedure ***ReadHexByte*** to input from the keyboard a byte ***b*** (i.e., two digits) and store ***b*** in the register AL. In order to achieve this, you must read consecutively two hexadecimal digits from the keyboard (use `ReadChar` for each digit). We assume that the user will enter only TWO hexadecimal

digits (i.e. '0' to 'F'). The first digit entered will be set as the most significant nibble of AL and the second digit entered will be set as the least significant nibble of AL. Process appropriately the two characters and store in AL the value meant by the user.

Examples:

1) Suppose the user enters the digit '9' followed by the digit '4'. This means that the user means the byte 94h. The procedure ***ReadHexByte*** must ultimately store in AL the value 94h = $(1001\ 0100)_2$. The digit '9' 'produced the most significant nibble $(1001)_2$ and the digit '4' 'produced the least significant nibble $(0100)_2$.
2) Suppose the user enters the digit 'B' followed by the digit '8'. This means that the user means the byte B8h. The procedure ***ReadHexByte*** must ultimately store in AL the value B8h = $(1011\ 1000)_2$. The digit 'B' 'produced the most significant nibble $(1011)_2$ and the digit '8' 'produced the least significant nibble $(1000)_2$.

Programming Exercise 5 (8 points): (Implementing ***SumFirstN*****)**

The objective is to write the procedure ***SumFirstN*** to compute the sum ***S*** of the first n integers (1 + 2 + ... + n) and to store ***S*** in the register DX. The procedure ***SumFirstN*** uses the register AL as a parameter. AL contains the number n. The constraint is that you must use only the sum instruction (ADD) to implement this procedure: do not use multiplication or division.

Examples:

1) If AL = 08h, then DX ← S = 1 + 2 +… + 7 + 8.
2) If AL = 3Eh, then DX ← S = 1 + 2 +… + 3Ch + 3Dh + 3Eh.

Programming Exercise 6 (8 points): (Program using the procedures used**)**

The objective of this exercise is to write the main program to:

1) Read from the keyboard a number ***N*** (Use ***ReadHexByte***)
2) Compute the sum S = 1 + 2 + 3 + 4 ... + N (Use ***SumFirstN***)
3) Display on the console the sum ***S***.

What you need to turn in:

- Electronic copy of your report (standalone) and source code (zipped) of your program. The five exercises must be included in the same project. Your project folder must be zipped folder named ***m4-name***, where *name* is your lastname. Zip the project folder and post it with the standalone report on Canvas. Submit separately (not inside the zipped folder) the report as a Microsoft Word or PDF file.
- Your report must:
  - State whether your code works.
  - Clearly explain how to compile and execute your code.
  - If needed/applicable, report/analyze (as appropriate) the results. The quality of analysis and writing is critical to your grade.
- Good writing and presentation are expected.

How this assignment will be graded:

The program compiles and executes correctly without any apparent bugs. The code is well-designed and commented.	100% credit
The program compiles and executes with minor bugs. The code is well-designed and commented.	90% credit
The program compiles and executes with major bugs.	40% credit
The program compiles but does not produce any meaningful result.	5% credit

If the instructor needs additional communications/actions to compile and execute your code, then a 30% penalty will be applied. If the turn-in instructions are not correctly followed, 10 points will be deducted.