

COMSC 260

Fall 2020

Programming Assignment 5

Worth 15 points (1.5% of your grade)

**DUE: Tuesday, 10/13/20 by 11:59 P.M. on
Canvas**





NOTE: You need to start by downloading the 260_assign5.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_assign5.asm** and **firstname_lastname_260_assign5.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_assign5.asm
James_Smith_260_assign5.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.).

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. Each sample run needs to use different inputs for the variables var1 – var4, and 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.). All of your sample runs should follow this format – for each individual sample run, screenshot the variables var1 – var4 in the data segment **AND** the corresponding values in EAX and EBX in the registers pane (once the program is finished running):

```
.data
    var1 BYTE 'A'
    var2 BYTE 'B'
    var3 BYTE 'C'
    var4 BYTE 'D'
```

Watch 1 ▢ ✕	
Name	Value
 var1	0x41 'A'
 var2	0x42 'B'
 var3	0x43 'C'
 var4	0x44 'D'

EAX = 44414243

EBX = BCBFBEBD

For this programming assignment you should **ONLY** use these instructions:

- 1) mov (regular/standard move)
- 2) add
- 3) sub (the sub instruction, like mov and add, also has two operands: a destination and a source)

For example: sub ah, al (ah is the destination, al is the source; this does the subtraction $ah - al$, and stores the result in ah. Register al is NOT modified by this instruction).

Operands for instructions should be registers or memory locations. An exception is that you can use the immediate values 0, 1, -1, 0Fh, or 0FFh where it would make sense.

The .data segment should NOT be modified.

The program has the following initial state in memory:

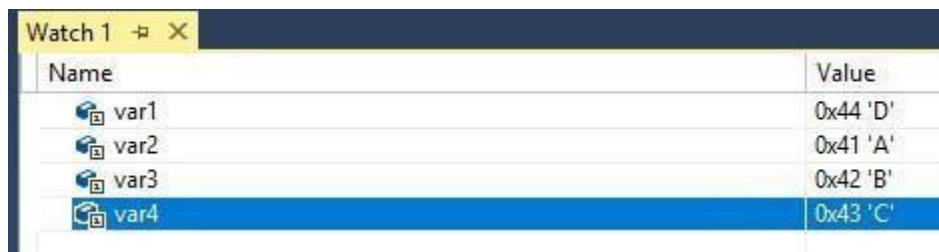
```
.data
var1 BYTE 'A'
var2 BYTE 'B'
var3 BYTE 'C'
var4 BYTE 'D'
```

Watch 1	
Name	Value
var1	0x41 'A'
var2	0x42 'B'
var3	0x43 'C'
var4	0x44 'D'

PHASE 1:

For phase 1 of the program, you need to **SHIFT** the contents of the four memory locations such that var1 will be storing 'D' (44h), var2 will be storing 'A' (41h), var3 will be storing 'B' (42h), and var4 will be storing 'C' (43h).

After completing phase 1, your watch window should look like this:



The screenshot shows a debugger's Watch window titled 'Watch 1'. It contains a table with two columns: 'Name' and 'Value'. There are four rows, each representing a variable: var1, var2, var3, and var4. var1 has a value of 0x44 'D', var2 has 0x41 'A', var3 has 0x42 'B', and var4 has 0x43 'C'. The row for var4 is highlighted in blue.

Name	Value
var1	0x44 'D'
var2	0x41 'A'
var3	0x42 'B'
var4	0x43 'C'

Phase 1 is approximately nine (9) instructions

PHASE 2:

For phase 2 you need store the contents of var1, var2, var3, and var4 altogether in register eax. More specifically, var1 will be stored in the highest byte of eax, var2 will be stored in the second highest byte, var3 will be stored in the second lowest byte, and var4 will be stored in the lowest byte.

You cannot directly refer to the highest 16 bits of eax, but you can refer to the lowest 16 bits with register ax, and the second lowest byte (8 bits) with register ah, and the lowest byte with register al. Since you cannot directly move anything into the upper 16 bits of eax, you need to think about how you could go about moving var1 and var2 into the lower 16 bits of eax, and then **shift** them into the upper 16 bits of eax. Then once the shifting is done, you could directly move var3 and var4 into their appropriate positions in the lower 16 bits of eax.

NOTE: Remember when you are dealing with byte variables, you need to work with 8-bit registers (ah, al, etc.)

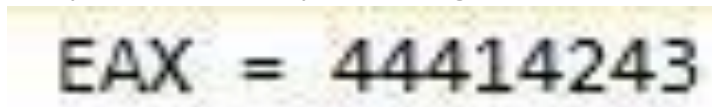
HINT: Think about how **add** instructions could be used to achieve shifting. For example, suppose we have the following 8-bit binary number:

000000**11**

After shifting this binary number two times to the left, we have:

0000**11**00

After phase 2 is completed, register eax should look like this:



(which means eax = “DABC”)

Phase 2 is approximately twenty-one (21) instructions

PHASE 3:

For phase 3 you need to store the **two’s complement** of var1, var2, var3, and var4 in register ebx. The two’s complement of var1 is in the highest byte of ebx, the two’s complement of var2 is the next byte, the two’s complement of var3 is in the next byte, and the two’s complement of var4 is in the lowest byte. Remember that for a hexadecimal number, there are two steps to get the two’s complement:

- 1) subtract the number from 15
- 2) Add 1

For example, the 'D' (44h) is stored in the highest byte of eax, so the two's complement of 44h will be stored in the highest byte of ebx. The two's complement of 44h is:

1) $15 - 4 = B$ (11) and

$15 - 4 = B$ (11)

So we get BBh after step 1

2) $BB + 1 = BC$

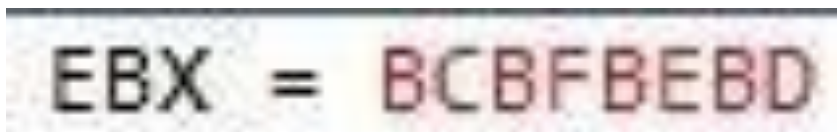
So the two's complement of 44h is BCh, and this is what should end up in the highest byte of ebx

HINT: Think about how the **sub** instruction could be used to get the two's complement of a number.

NOTE: You should use the **same shifting algorithm** that you used for phase 2 here. Start by placing the **two's complement** of var1 and var2 into the lower

16 bits of ebx, and then shift them into the upper 16 bits of ebx. Once that is completed, directly move the **two's complement** of var3 and var4 to the lower 16 bits of ebx.

After phase 3 is completed, register ebx should look like this:



Phase 3 is approximately twenty-one (21) instructions

PROGRAM FINAL OUTPUTS

To recap, this is what you should have after the entire program is finished:

Watch 1	
Name	Value
var1	0x44 'D'
var2	0x41 'A'
var3	0x42 'B'
var4	0x43 'C'

EBX = BCBFBEBD

EBX = BCBFBEBD

Phase 1 is approximately nine (9) instructions

Phase 2 is approximately twenty-one (21) instructions

Phase 3 is approximately twenty-one (21) instructions