

COMSC 260

Fall 2020

Programming Assignment 6

Worth 15 points (1.5% of your grade)

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

NOTE: You need to start by downloading the 260_assign6.asm file from the Programming Assignment 6 folder on 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_assign6.asm** and **firstname_lastname_260_assign6.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_assign6.asm James_Smith_260_assign6.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.). **Important Note: Any of my comments in the program that you download from Canvas 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. Each sample run needs to use different inputs for the registers ax, bx, si, di, sp, and bp, and your sample runs should **NOT** be the same as the sample runs that are used in this write-up for the assignment. Each of the six registers needs to store two characters.

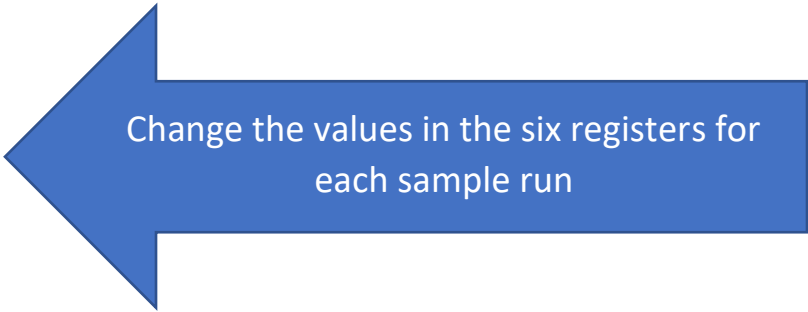
You should also number each of your sample runs (i.e., sample run 1, sample run 2, etc.). All of your sample runs should follow this format – for each individual sample run, screenshot (1) the values used in the registers (ax, bx, si, di, sp and bp) at the beginning of the program, (2) the values in the msgforward and msgbackward arrays at the end of the program, and (3) the values in the registers eax, ebx, esi, edi, esp, and ebp at the end of the program. For example:

(1)

```
.code
main proc

    mov eax, 0
    mov ebx, 0
    mov ecx, 0
    mov edx, 0
    mov esi, 0
    mov edi, 0
    mov esp, 0
    mov ebp, 0

    mov ax, "EM"
    mov bx, "OH"
    mov si, "E"
    mov di, "MO"
    mov sp, "CL"
    mov bp, "EW"
```



Change the values in the six registers for each sample run

(2)

For msgforward array (starts at address 00404000):

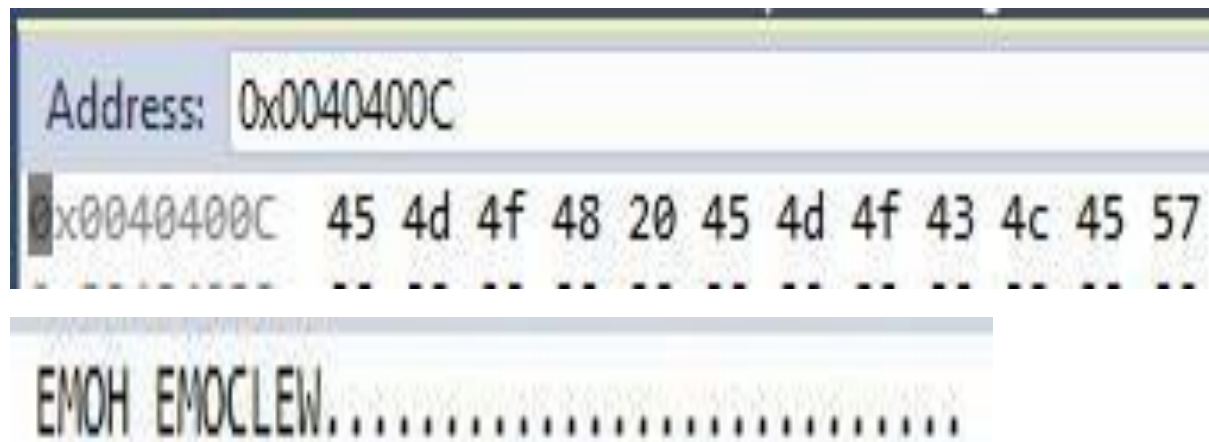


Address: 0x00404000

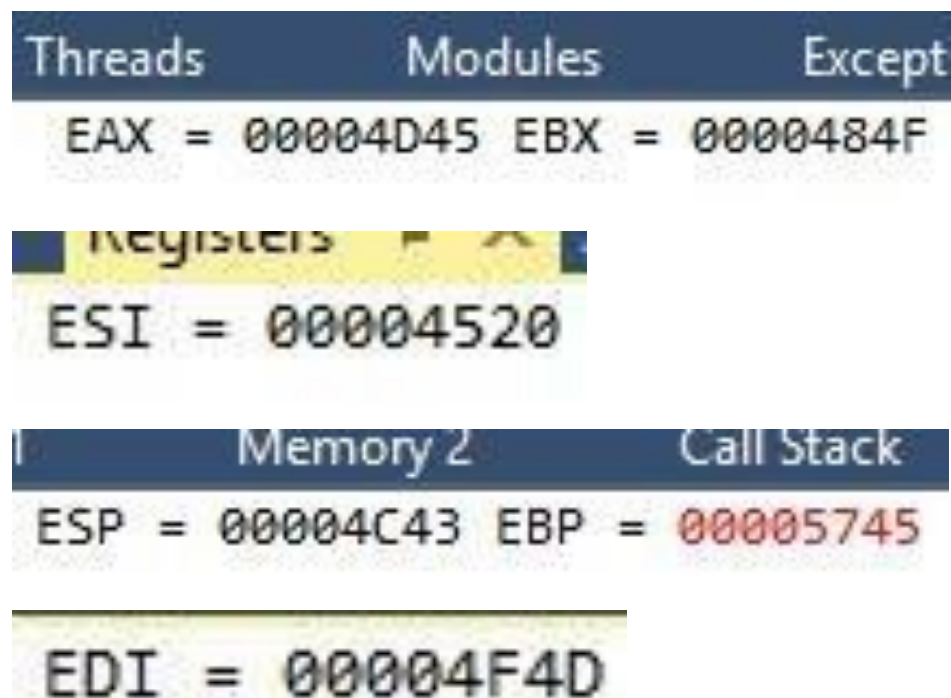
0x00404000	57	45	4c	43	4f	4d	45	20	48	4f	4d	45
------------	----	----	----	----	----	----	----	----	----	----	----	----

WELCOME HOME.....

For msgbackward array (starts at address 0040400C)



(3)



For this programming assignment you should **ONLY** use the **mov** instruction (standard/regular move)

Operands for instructions should be registers or memory locations. An exception is that you can use the immediate value 0 if you need to clear a register. **NO other immediate/literal values should be used in your instructions.**

The .data segment should NOT be modified.

The program has the following initial state in memory:

```
.data
msgforward WORD 6 DUP(?)
msgbackward WORD 6 DUP(?)
```

And you are given the following initial instructions:

```
.code
main proc

    mov eax, 0
    mov ebx, 0
    mov ecx, 0
    mov edx, 0
    mov esi, 0
    mov edi, 0
    mov esp, 0
    mov ebp, 0

    mov ax, "EM"
    mov bx, "OH"
    mov si, "E"
    mov di, "MO"
    mov sp, "CL"
    mov bp, "EW"
```

And the registers initially look like this:

EAX = 0000454D EBX = 00004F48

ESI = 00002045

ESP = 0000434C EBP = 00004557

EDI = 00004D4F

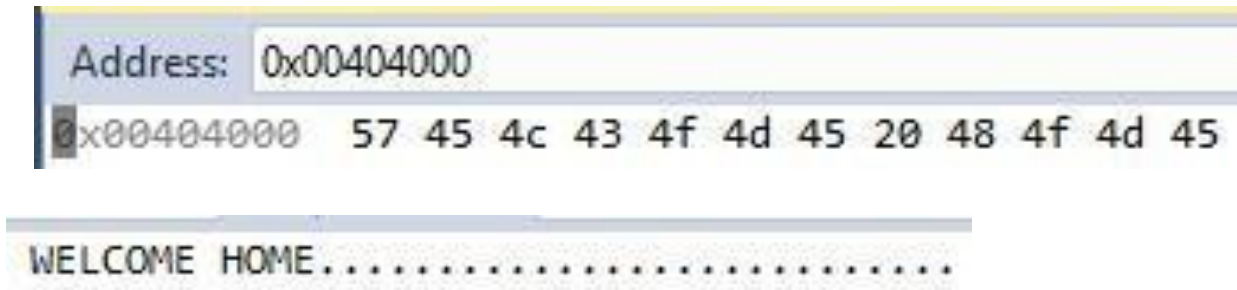
The message “Welcome Home”, which contains a total of 12 bytes, is being stored in reverse order across registers ax, bx, si, di, sp, and bp. Notice that each of the 16-bit registers is storing two letters of the message.

PHASE 1:

Phase 1 is approximately six (6) instructions

For phase 1, you need to move the message that is being stored across the six registers in forward order to the msgforward word array. This means after phase 1 is completed, the msgforward array should be storing “Welcome Home”.

If you type in &msgforward in the memory window, your memory should look like this:



And your watch window should look like this:

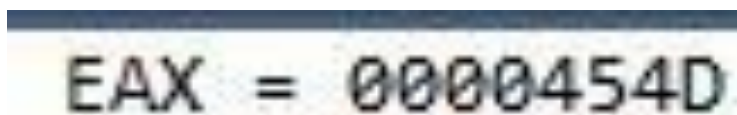
Watch 1	
Name	Value
msgforward	0x4557
msgbackward	0x0000

Notice that the watch window only shows hex values for the first two letters of the message, “WE”, which are 57h and 45h respectively. This is why the memory window needs to be used to see what is going on in memory.

PHASE 2:

Phase 2 is approximately twenty-two (22) instructions

For phase two you need to reverse the ordering of each of the two letters in each of the six registers. For example, initially register ax is storing “EM”, where the “E” (45h) is in ah and the “M” (4D) is in al:



Your goal is to reverse the order, so that the “M” (4D) is in ah and the “E” (45h) is in al. The same needs to be done for the other five registers as well.

After phase 2 is completed, the six registers should look like this:

EDI = 00004F4D

Memory 2 Call Stack
ESP = 00004C43 EBP = 00005745

Memory 2 Call Stack
ESP = 00004C43 EBP = 00005745

EDI = 00004F4D

NOTE: Registers **cx** and **dx** are unused and are available to help with the swapping.

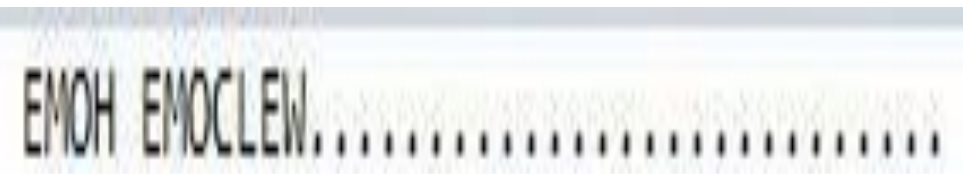
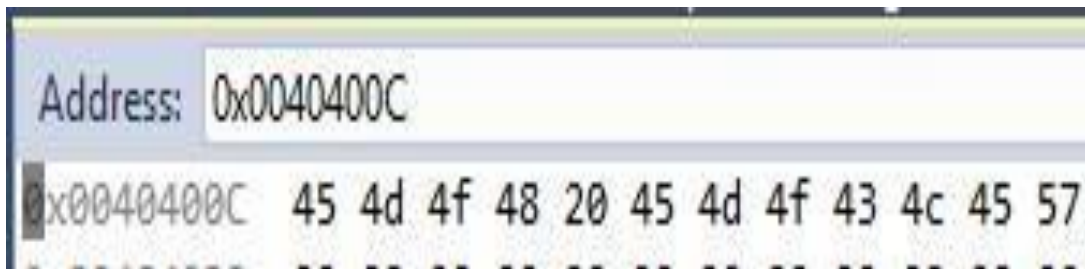
PHASE 3:

Phase 3 is approximately six (6) instructions



For phase 3, you need to move the message that is being stored across the six registers in backward order to the msgbackward word array. This means after phase 1 is completed, the msgbackward array should be storing

“EMOH EMOCLEW” (WELCOME HOME backwards)

After phase 3 is completed, your memory should look like this if you type in &msgbackward:



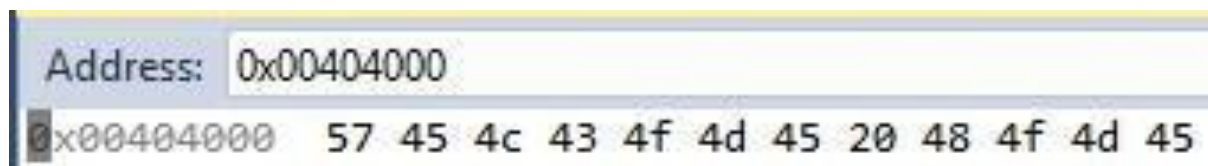
And your watch window should look like this:

Watch 1 ▢ ✕	
Name	Value
 msgforward	0x4557
 msgbackward	0x4d45

PROGRAM FINAL OUTPUTS

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

For &msgforward



WELCOME HOME.....

EDI = 00004F4D

Registers

ESI = 00004520

Memory 2

Call Stack





ESP = 00004C43 EBP = 00005745

EDI = 00004F4D

For &msgbackward

Address:	0x0040400C
0x0040400C	45 4d 4f 48 20 45 4d 4f 43 4c 45 57

EMOH EMOCLEW.....

Watch 1  	
Name	Value
 msgforward	0x4557
 msgbackward	0x4d45