COMSC 260

Fall 2020

Programming Assignment 4

Worth 15 points (1.5% of your grade)

DUE: Tuesday, 10/6/20 by 11:59 P.M. on Canvas

Start by downloading the following two (2) files from the Programming Assignment 4 folder on Canvas**:**

260_assign4_prog1.asm

260_assign4_prog2.asm

NOTE: Your submission for this assignment should be **two .asm files (see above)**.

The following naming convention should be used for naming your **program 1 file**: **firstname_lastname_260_assign5_prog1.asm**

The following naming convention should be used for naming your **program 2 file**: **firstname_lastname_260_assign5_prog2.asm**

For example, if your first name is “James” and your last name is “Smith”, then your two files would be named **James_Smith_260_assign5_prog1.asm and James_Smith_260_assign5_prog2.asm**

Problem 1) Worth 7.5 points (0.75% of your grade)

Suppose we have the following C++ program:

prog1.cpp

```cpp
#include <iostream>
using namespace std;


int main()
{

    int A, B=5, C=10, D=4, E=6, F=7, G=8, H=2, I=3, J=12;


     A = B+C + D*B + B*E + E*F + F*G + D*D + B*B + H*I + I*D + H+J;

     cout<<A<<endl<<endl;

    system("PAUSE");
    return 0;


}
```

For your first programming problem, you will be implementing the above program in **<u>x86 assembly language</u>** with the following constraint: the **only** instruction that can be used is the **add** instruction

The idea is that multiplication is really just repeated addition. For example, 5*4 can be represented as either 5+5+5+5 or 4+4+4+4+4. The former is the more ideal representation, as it requires less addition operations to be performed. In other words, the larger number should be added to itself the shorter number of times.

Under the Programming Assignment 4 folder on Canvas, download the **260_assign5_prog1.asm** file. This file already has some code in it, so be sure to use it as your starting point:

assign2_prog1.asm Disassembly 260prog1asm.asm

```
; Program template

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.data
    A DWORD ?
    B DWORD 5
    C1 DWORD 10
    D DWORD 4
    E DWORD 6
    F DWORD 7
    G DWORD 8
    H DWORD 2
    I DWORD 3
    J DWORD 12


.code
main proc
mov eax, 0

; INSERT ALL ADD INSTRUCTIONS HERE
; INSERT ALL ADD INSTRUCTIONS HERE
; INSERT ALL ADD INSTRUCTIONS HERE
; INSERT ALL ADD INSTRUCTIONS HERE
; INSERT ALL ADD INSTRUCTIONS HERE

mov A, eax


    invoke ExitProcess,0
main endp
end main
```

100 %

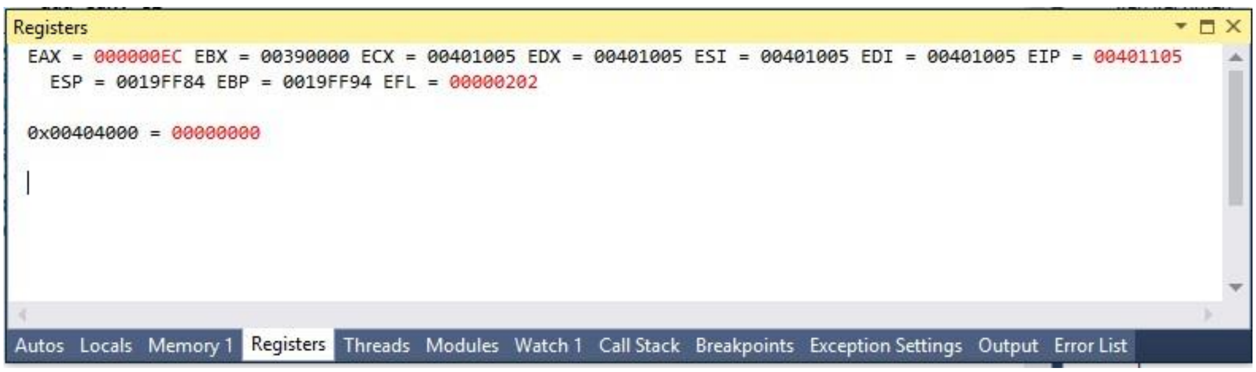
Insert all of your add instructions in-between the two mov instructions. You should have approximately ~**40 add instructions** in total.

The **operands** for all add instructions should be **register EAX** or **variables** from the data segment.

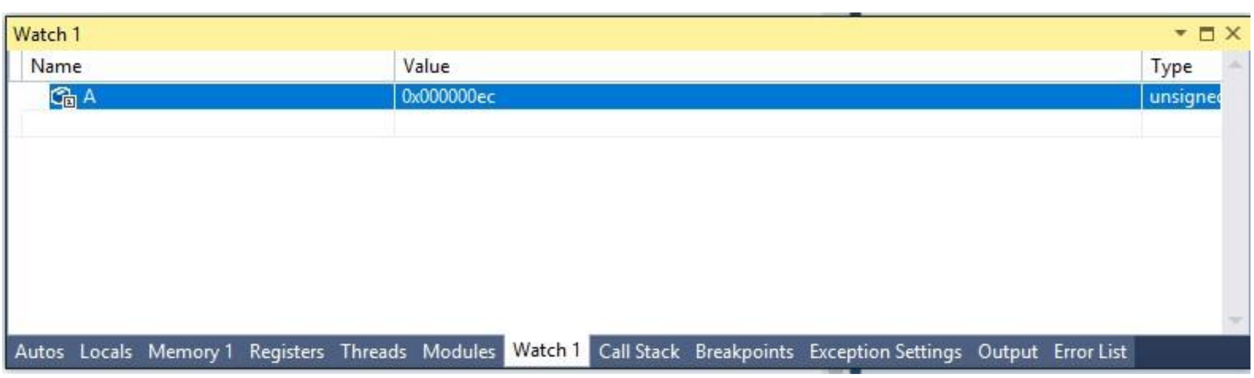
Example: add eax, B

eax is the only register that needs to be used for this problem.

After you execute the ~40 add instructions, your registers should look something like this:



In particular, the register EAX should be storing the 32-bit hex value **000000EC**, which is 236 decimal (16*14 + 12 = 236)



Memory location A should now also be storing **000000EC,** because the mov A, eax instruction at the very end of the program copies the contents of register eax into the memory location represented by the variable A.

Problem 2) Worth 7.5 points (0.75% of your grade)

Suppose we have the following C++ program:

prog2.cpp

```cpp
#include <iostream>
using namespace std;


int main()
{
    int b=1, c=2, d=3, e=4, f=5, g=6, h=7, i=8, j=9, k=10, l=11, m=12, n=13, o=14, p=15;

    int a = b + c + d * e * f * g * h * i * j * k * l + m + n + o + p;

    cout<<a<<endl<<endl;

    system("PAUSE");
    return 0;


}
```

For your second programming problem, you will be implementing the above program in **x86 assembly language** with the following constraint: the **only** instruction that can be used is the **add** instruction

---

As with the first problem, the idea is that you will be representing all multiplications as repeated additions.

Under the Programming Assignment 4 folder on Canvas, download the **260_assign4_prog2.asm** file. This file already has some code in it, so be sure to use it as your starting point:

assign2_prog2.asm | Disassembly | 260prog2.asm | assign2_prog1.asm

```

; Program template

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.data

A DWORD ?
B DWORD 1
C1 DWORD 2  ; C is a reserved word
D DWORD 3
E DWORD 4
F DWORD 5
G DWORD 6
H DWORD 7
I DWORD 8
J DWORD 9
K DWORD 10
L DWORD 11
M DWORD 12
N DWORD 13
O DWORD 14
P DWORD 15

.code
main proc

mov eax, 0 ; initialize eax to 0

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=12

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

```

```
mov ebx, eax ; ebx=60

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=360

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=2520

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=20,160

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=181,440

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov ebx, eax ; ebx=1,814,400

; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE
; INSERT ADD INSTRUCTIONS HERE

mov A, eax

    invoke ExitProcess,0
main endp
end main
```

Insert all of your add instructions in-between all of the mov instructions. You should have approximately ~**58 add instructions** in total.

You will need to use **two** registers for this problem: **eax** and **ebx**

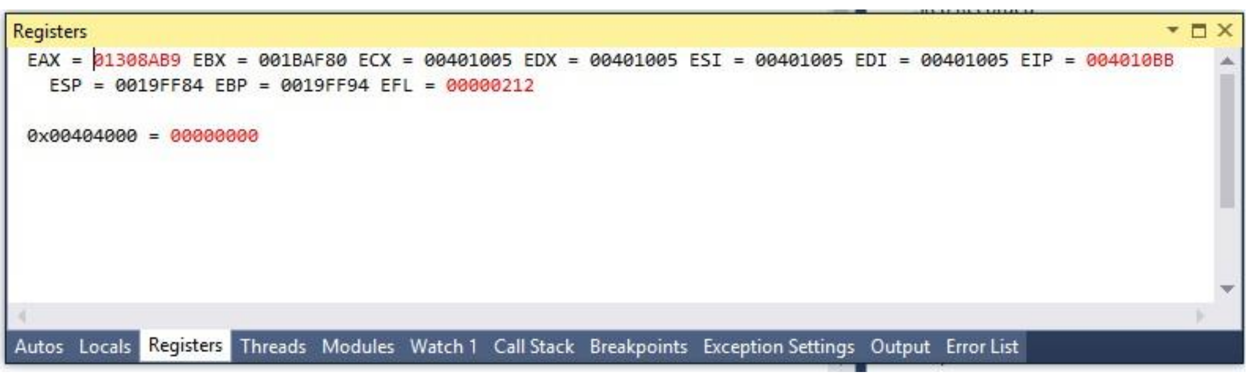
The operands for all add instructions should be **registers** (**EAX** or **EBX**) or **variables** from the data segment. Examples:

add eax, B

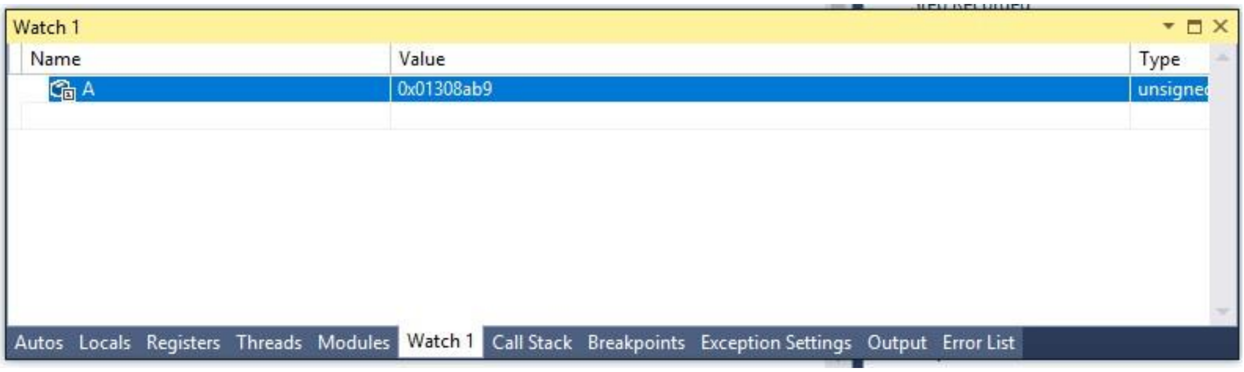
and

add eax, ebx

After you execute the ~58 add instructions, your registers should look something like this:



In particular, the register EAX should be storing the 32-bit hex value **01308AB9**, which is 19,958,457 decimal ($16^6 + 16^5*3 + 16^3*8 + 16^2*10 + 16*11 + 9 =$ 19,958,457)



Memory location A should now also be storing **01308AB9,** because the mov A, eax instruction at the very end of the program copies the contents of register eax into the memory location represented by the variable A.
