

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)

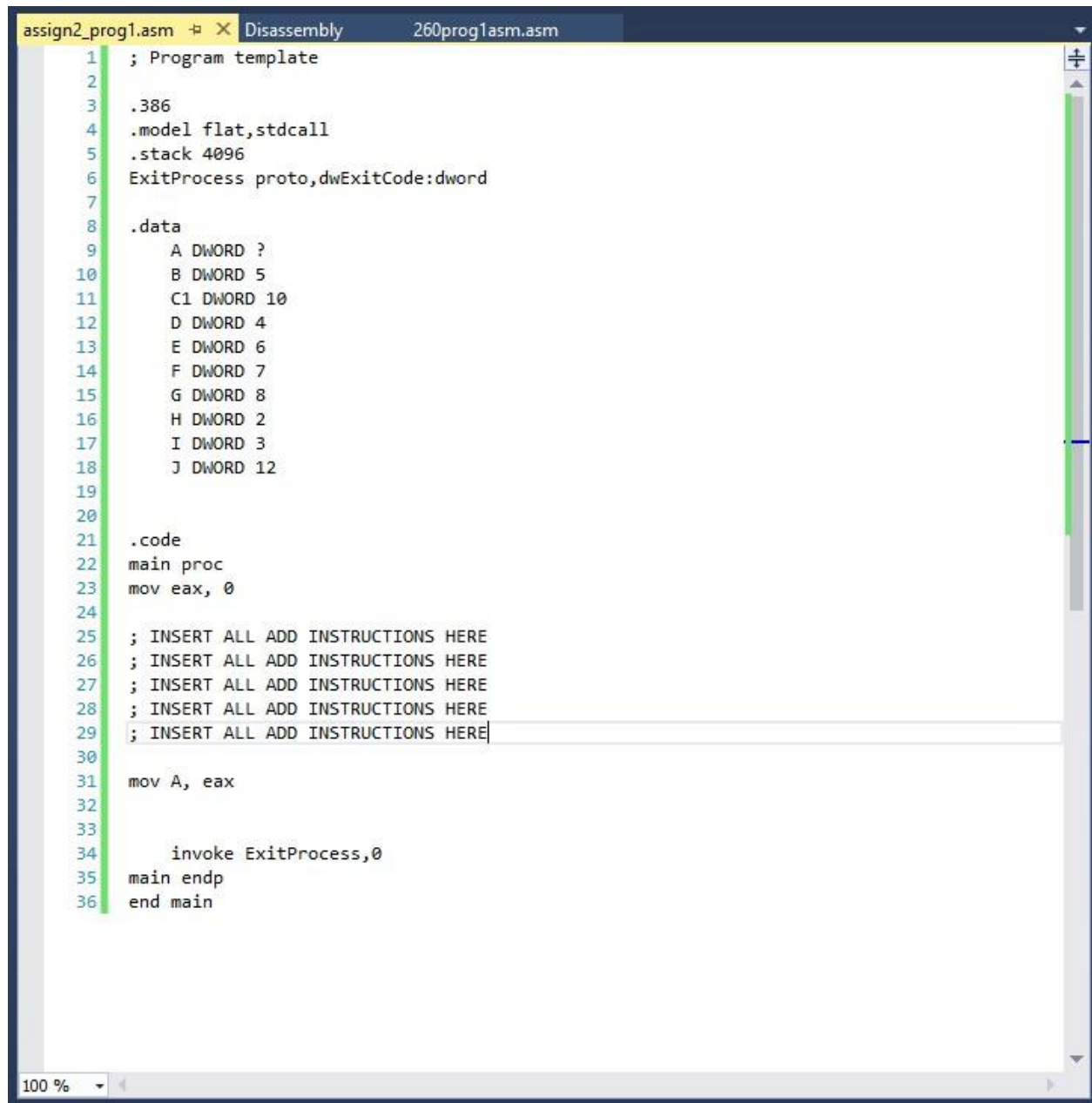
Suppose we have the following C++ program:

```
prog1.cpp
1  #include <iostream>
2  using namespace std;
3
4
5  int main()
6  {
7
8      int A, B=5, C=10, D=4, E=6, F=7, G=8, H=2, I=3, J=12;
9
10
11     A = B+C + D*B + B*E + E*F + F*G + D*D + B*B + H*I + I*D + H+J;
12
13     cout<<A<<endl<<endl;
14
15     system("PAUSE");
16     return 0;
17
18
19 }
```

For your first 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

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:



```
1 ; Program template
2
3 .386
4 .model flat,stdcall
5 .stack 4096
6 ExitProcess proto,dwExitCode:dword
7
8 .data
9     A DWORD ?
10    B DWORD 5
11    C1 DWORD 10
12    D DWORD 4
13    E DWORD 6
14    F DWORD 7
15    G DWORD 8
16    H DWORD 2
17    I DWORD 3
18    J DWORD 12
19
20
21 .code
22 main proc
23     mov eax, 0
24
25     ; INSERT ALL ADD INSTRUCTIONS HERE
26     ; INSERT ALL ADD INSTRUCTIONS HERE
27     ; INSERT ALL ADD INSTRUCTIONS HERE
28     ; INSERT ALL ADD INSTRUCTIONS HERE
29     ; INSERT ALL ADD INSTRUCTIONS HERE
30
31     mov A, eax
32
33
34     invoke ExitProcess,0
35 main endp
36 end main
```

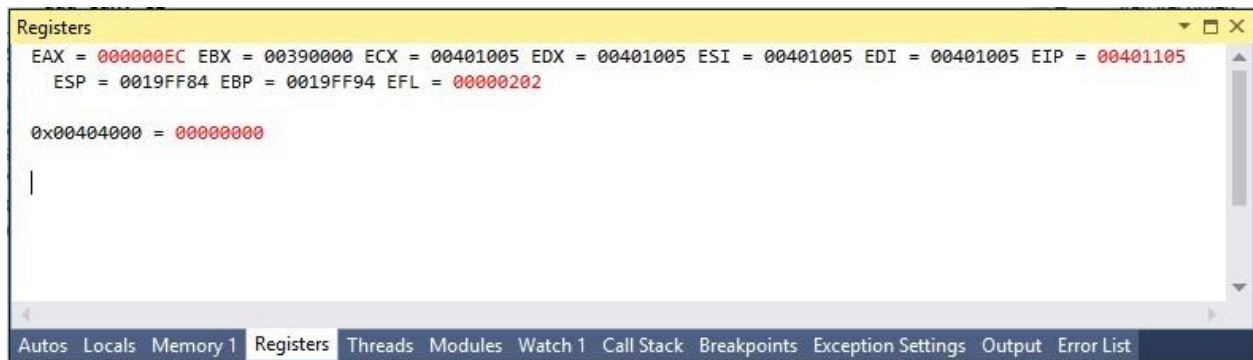
Insert all of your add instructions in-between the two mov instructions. You should have approximately **~40 add instructions** in total.

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

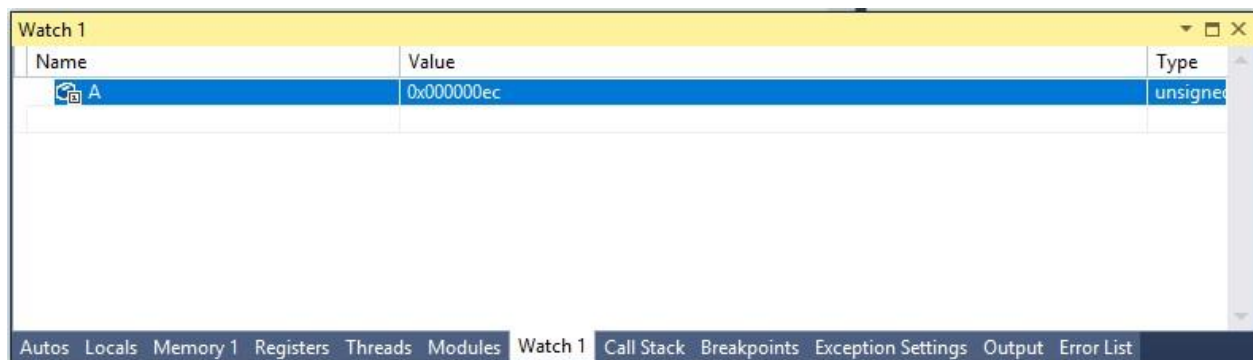
Example: add eax, B

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 ($16 \times 14 + 12 = 236$)



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
1  #include <iostream>
2  using namespace std;
3
4
5  int main()
6  {
7      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;
8
9      int a = b + c + d * e * f * g * h * i * j * k * l + m + n + o + p;
10
11     cout<<a<<endl<<endl;
12
13     system("PAUSE");
14     return 0;
15
16
17 }
18
```

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:

```
1
2 ; Program template
3
4 .386
5 .model flat,stdcall
6 .stack 4096
7 ExitProcess proto,dwExitCode:dword
8
9 .data
10
11 A DWORD ?
12 B DWORD 1
13 C1 DWORD 2 ; C is a reserved word
14 D DWORD 3
15 E DWORD 4
16 F DWORD 5
17 G DWORD 6
18 H DWORD 7
19 I DWORD 8
20 J DWORD 9
21 K DWORD 10
22 L DWORD 11
23 M DWORD 12
24 N DWORD 13
25 O DWORD 14
26 P DWORD 15
27
28 .code
29 main proc
30
31 mov eax, 0 ; initialize eax to 0
32
33 ; INSERT ADD INSTRUCTIONS HERE
34 ; INSERT ADD INSTRUCTIONS HERE
35 ; INSERT ADD INSTRUCTIONS HERE
36
37 mov ebx, eax ; ebx=12
38
39 ; INSERT ADD INSTRUCTIONS HERE
40 ; INSERT ADD INSTRUCTIONS HERE
41 ; INSERT ADD INSTRUCTIONS HERE
42
```

```
assign2_prog2.asm  X Disassembly  260prog2.asm  assign2_prog1.asm
43  mov ebx, eax ; ebx=60
44
45  ; INSERT ADD INSTRUCTIONS HERE
46  ; INSERT ADD INSTRUCTIONS HERE
47  ; INSERT ADD INSTRUCTIONS HERE
48
49  mov ebx, eax ; ebx=360
50
51  ; INSERT ADD INSTRUCTIONS HERE
52  ; INSERT ADD INSTRUCTIONS HERE
53  ; INSERT ADD INSTRUCTIONS HERE
54
55  mov ebx, eax ; ebx=2520
56
57  ; INSERT ADD INSTRUCTIONS HERE
58  ; INSERT ADD INSTRUCTIONS HERE
59  ; INSERT ADD INSTRUCTIONS HERE
60
61  mov ebx, eax ; ebx=20,160
62
63  ; INSERT ADD INSTRUCTIONS HERE
64  ; INSERT ADD INSTRUCTIONS HERE
65  ; INSERT ADD INSTRUCTIONS HERE
66
67  mov ebx, eax ; ebx=181,440
68
69  ; INSERT ADD INSTRUCTIONS HERE
70  ; INSERT ADD INSTRUCTIONS HERE
71  ; INSERT ADD INSTRUCTIONS HERE
72
73  mov ebx, eax ; ebx=1,814,400
74
75  ; INSERT ADD INSTRUCTIONS HERE
76  ; INSERT ADD INSTRUCTIONS HERE
77  ; INSERT ADD INSTRUCTIONS HERE
78
79  mov A, eax
80
81      invoke ExitProcess,0
82  main endp
83  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

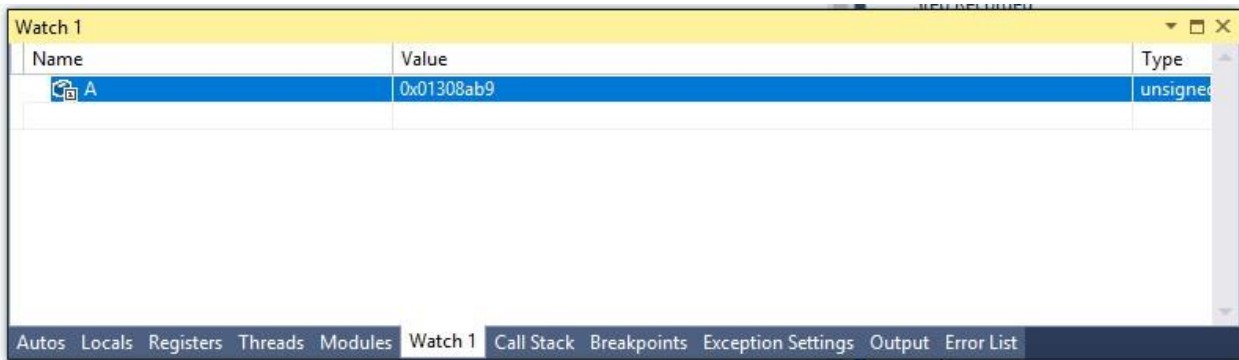
and

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 ($16^6 + 16^5 \cdot 3 + 16^3 \cdot 8 + 16^2 \cdot 10 + 16 \cdot 11 + 9 = 19,958,457$)



The screenshot shows a debugger's 'Watch' window. The window has a yellow title bar labeled 'Watch 1'. Below the title bar is a table with three columns: 'Name', 'Value', and 'Type'. The first row of the table is highlighted in blue and contains the variable 'A' in the 'Name' column, the hexadecimal value '0x01308ab9' in the 'Value' column, and the text 'unsigned' in the 'Type' column. Below the table is a large empty white area. At the bottom of the window is a dark blue tab bar with several tabs: 'Autos', 'Locals', 'Registers', 'Threads', 'Modules', 'Watch 1' (which is selected), 'Call Stack', 'Breakpoints', 'Exception Settings', 'Output', and 'Error List'.

| Name | Value | Type |
|------|------------|----------|
| A | 0x01308ab9 | unsigned |

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.
