COMSC 260

Fall 2020

Programming Assignment 8

Worth 15 points (1.5% of your grade)

DUE: Tuesday, 11/3/20 by 11:59 P.M. on Canvas

START by downloading the assign8.asm file from the Programming Assignment 8 folder on Canvas

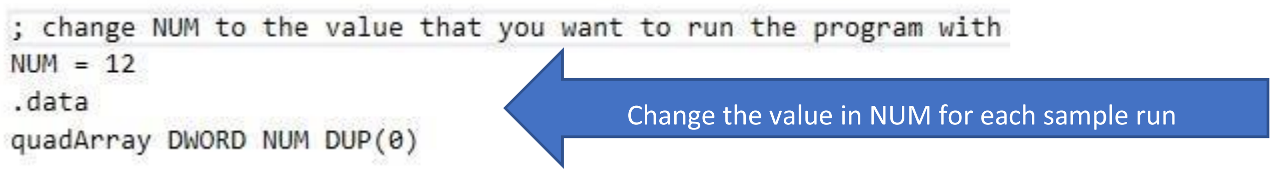
NOTE: Your submission for this assignment should be a single **.asm** file and a single **.pdf** file. The following naming convention should be used for naming your files: **firstname_lastname_260_assign8.asm and firstname_lastname_260_assign8.pdf**. The pdf file that you submit should contain the screenshots 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_assign8.asm and James_Smith_260_assign8.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.). **NOTE: My comments 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 NUM constant, 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 (1) the value used in the NUM constant and (2) the values in the quadArray array at the end of the program. For example:

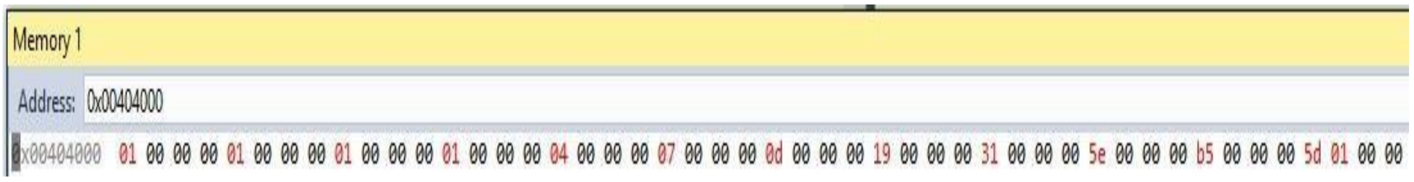
(1) Value in the NUM constant



```
; change NUM to the value that you want to run the program with
NUM = 12
.data
quadArray DWORD NUM DUP(0)
```

Note: Do NOT change quadArray directly in the data segment!!! The size of quadArray is determined by NUM, and initially each location in the array will store 0. For example in the screenshot shown above, the quadArray would initially store the value 0 twelve times. Your program logic will build the quadArray with the correct values using a loop.

(2) Values in the quadArray array (after the array has been built via a loop)



```
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 07 00 00 00 0d 00 00 00 19 00 00 00 31 00 00 00 5e 00 00 00 b5 00 00 00 5d 01 00 00
```

For your programming assignment this week you will be writing an x86 assembly program that will calculate a modified version of the Fibonacci sequence, called "Quadonacci". **You will be building the quadArray with the correct values using a loop (i.e., the loop instruction)**. A loop is needed as your program needs to build the correct quadArray for **ANY** value that is stored in NUM. The idea with Quadonacci is that you have four base cases:

Quad(1) = 1

Quad(2)= 1

Quad(3) = 1

Quad(4) = 1

And for all N >=5,

Quad(N) = Quad(N-1) + Quad(N-2) + Quad(N-3) + Quad(N-4)

For example:

Quad(5) = Quad(4) + Quad(3) + Quad(2) + Quad(1)

Quad(5) = 1+1+1+1 = 4

And

Quad(6) = Quad(5) + Quad(4) + Quad(3) + Quad(2)

Quad(6) = 4 + 1 + 1+ 1 = 7

And

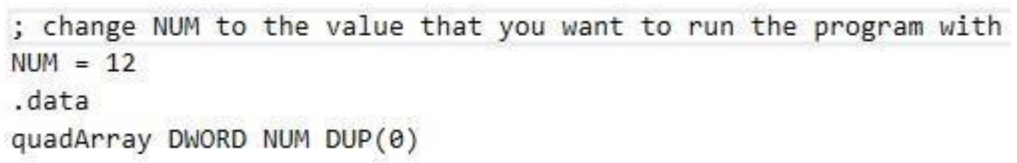
Quad(7) = Quad(6) + Quad(5) + Quad(4) + Quad(3)

Quad(7) = 7 + 4 + 1 + 1 = 13 etc.

Here is a table showing the first twelve Quadonacci values:

Q(1)	Q(2)	Q(3)	Q(4)	Q(5)	Q(6)	Q(7)	Q(8)	Q(9)	Q(10)	Q(11)	Q(12)
1	1	1	1	4	7	13	25	49	94	181	349

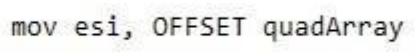
You are given the following declarations:



```
; change NUM to the value that you want to run the program with
NUM = 12
.data
quadArray DWORD NUM DUP(0)
```

The program should store each individual quad(i) number in the quadArray array in memory. Each individual quad(i) number is stored as a DWORD. Each DWORD in the quadArray is initialized to 0.

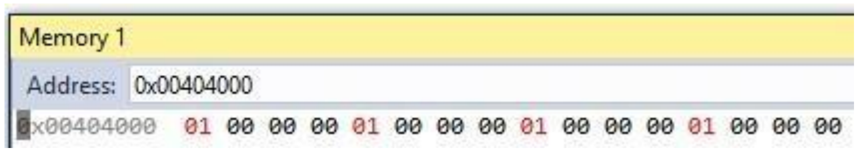
Furthermore, the **indirect operand** approach should always be used to access the quadArray array. You are given the following instruction in the code segment as a starting point:



```
mov esi, OFFSET quadArray
```

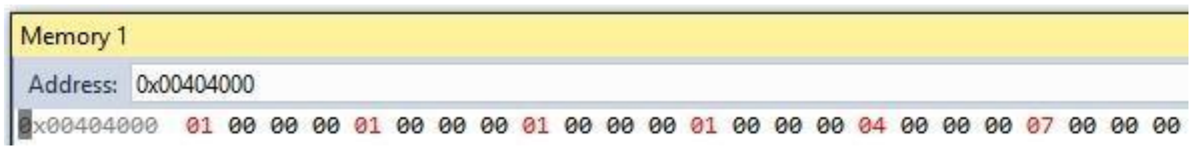
SAMPLE RUNS:

NUM = 4 (quadArray stores 1, 1, 1, 1 after the loop is finished)



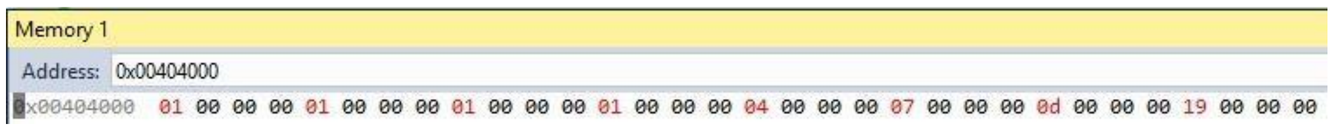
```
Memory 1
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
```

NUM = 6 (quadArray stores 1, 1, 1, 1, 4, 7 after the loop is finished)



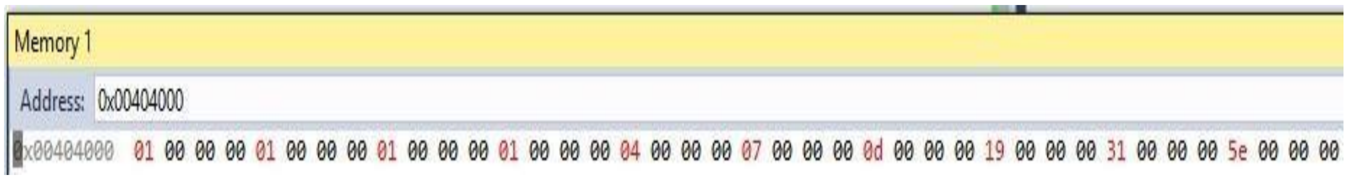
```
Memory 1
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 07 00 00 00
```

NUM = 8 (quadArray stores 1, 1, 1, 1, 4, 7, 13, 25 after the loop is finished)



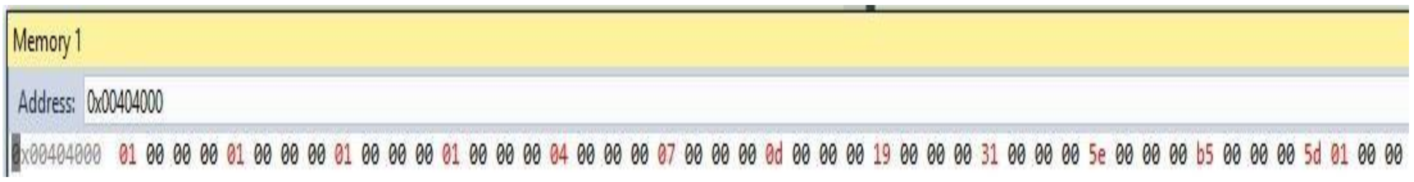
```
Memory 1
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 07 00 00 00 0d 00 00 00 19 00 00 00
```

NUM = 10 (quadArray stores 1, 1, 1, 1, 4, 7, 13, 25, 49, 94 after the loop is finished)



```
Memory 1
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 07 00 00 00 0d 00 00 00 19 00 00 00 31 00 00 00 5e 00 00 00
```

NUM = 12 (quadArray stores 1, 1, 1, 1, 4, 7, 13, 25, 49, 94, 181, 349 after the loop is finished)



```
Memory 1
Address: 0x00404000
0x00404000  01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 04 00 00 00 07 00 00 00 0d 00 00 00 19 00 00 00 31 00 00 00 5e 00 00 00 b5 00 00 00 5d 01 00 00
```